MASTER EN SCIENCES INFORMATIQUES

Didacticiel de mise en équation et de résolution d'un problème

Lefevre, Catherine

Award date:
1996

Awarding institution:
Universite de Namur

FACULTES UNIVERSITAIRES
NOTRE-DAME DE LA PAIX

NAMUR

Institut d'informatique
Rue Grandgagnage, 21 B
B-5000 Namur (Belgique)

Didacticiel de mise en équation et de résolution d'un problème

Catherine LEFEVRE

Mémoire réalisé en vue de l'obtention du grade de
Licenciée et Maître en Informatique

Promoteur : Monsieur Claude CHERTON

Année Académique 1995-1996

LBS 6844029

A Mes Parents

Remerciements

Ce mémoire n'a pu se réaliser que grâce à l'enthousiasme de plusieurs personnes que j'ai le plaisir de remercier.

Monsieur Claude Cherton, mon promoteur, professeur à l'institut d'informatique, qui m'a guidée tout au long de ce travail.

Messieurs Jean Donnay et Charles Duchâteau, professeurs au département pédagogique, pour leurs bons conseils.

Mesdames Joëlle Kok, Bernadette Lalot et Jacqueline Rekko, professeurs à l'institut Ilon-St Jacques, qui ont eu une part très active dans ce projet.

Je remercie également ma famille, tout particulièrement ma maman, qui m'a apporté toute sa compétence dans le domaine des mathématiques et ma petite soeur Delphine pour ses nombreux encouragements.

Un grand merci à Bernard, qui a mis tout son matériel informatique à ma disposition et m'a fait confiance jusqu'au bout.

Table des matières

Abstract

I have completed my studies by taking an aggregation's training. Within the framework of my dissertation, it has been possible for me to reconcile my computer knowledge with my pedagogical knowledge through a mathematical teachware.

The objective of my work is to help students understand the text of a mathematical problem, to transform the problem into a simple equation and to write the solution of the problem. This project addresses students aged 14-15 in secondary school.

I worked with a group of secondary school mathematics teachers. Then, I learned about teaching and learning of maths. The teaching of maths presents some crucial problems, for example the methodology, the role and emphasis of algebraic manipulation. I think that teachers had to be actively involved in the development of teachware.

The integration of my software into the educational environment doesn't redo the education's world. I just want to improve the quality of teaching. In relation to given didactic problems and teachers'need, I can, for example, minimize the time during which the teacher prepares his lessons. On the other hand, students can enjoy learning, in an interactive way and personalize their learning. They can discover on their own new mathematical subjects.

Teachware will not replace the teacher. 'Teaching' is a vocation, a teacher must be listening to his students, understand them, ... there is an emotional relation between a teacher and his students. A computer doesn't have feelings.

But the integration of computers into the educational environment include barriers to full access : a lack of personal preparation, the problem of unanticipated negative consequences of computer use, ... Teachers need to have new range of competencies in addition to their subject specialism, for example, the understanding of information technology. The computer may be seen as a mirror to help teachers reflect and revise the concepts of their role.

There are five important steps in writing and solving a simple equation. The different steps are 'Read the text', 'Choice the unknown', 'Transform the problem into a simple equation', 'Solve the equation', and 'Write the solution of the problem'.

Teachware is composed of two modules : the teacher's module and the student's module. Each module presents the different steps.

In the teacher's module, first the teacher writes the text of the problem. Then he selects any word in the text for giving an explanation. All explanations are gathered in a permanent database. If the meaning of a word differs from its usual meaning, it is stored in a special table. The teacher writes the equation of the problem and with the mouse, he can highlight each element or group of elements of the equation with one or more words of the text. The teacher's work is finished. Teachware solves the equation and gives the solution of the problem.

In the student's module, I provide help tools to the student. The first is called 'diagnostiqueur' and it permits student to correct his work, to write the equation, ... In fact, it is a dialog box in which the five steps are analyzed. Different icons represent the result of the analysis : a face with a smile means that the step is successful, a bomb means there is a mistake at this point, an 'up' arrow means that the previous step is unsuccessful and the student can't go on.

Following the analysis, three options are presented to the student. First, Teachware offers to guide the student, step by step. Second, it proposes to go on without any help (maybe the student understands his mistake). The last option refers to another help tool : It is a 'topics' aid. You 'click' on a highlight expression and the program gives you some explanations about this expression. Therefore, at any time, the student can use the topic research.

If a word in the text is not understood by the student, he can 'click' on it and Teachware gives him some explanations about it. In fact, Teachware creates a special table or database inquiry. The student can also type the word with the keyboard.

The same principle is used for highlighting the element or group of element of the student's equation with one or more words of the text.

Within this dissertation, I wanted to present teachware was not a mechanical tool, closing the door at teacher's involvement. It is important that teachers keep their status although their role is changing. I also wanted to explain that students can learn in another way, they can discover on their own lesson at their own pace. They can also set their learning goals. The integration of computer into the educational environment is not an easy thing but it seems to be the direction of future education. People must be prepared for not becoming future illiterates. If you use information technology intelligently, you are the winner of the next generation ...

Introduction générale

La présentation du sujet

Ce mémoire a pour thème la réalisation d'un didacticiel de mise en équation et de résolution d'un problème. Le premier objectif de ce travail est donc d'aider les élèves à comprendre l'énoncé en français d'un problème mathématique, à mettre ce problème en équation et, finalement, à calculer sa solution. Le second objectif consiste à présenter au professeur une nouvelle méthode pour préparer ses leçons.

Dans le domaine des mathématiques, de nombreux logiciels existent déjà, tels que le Cabri-géomètre, Abacus, etc. Le travail que nous avons effectué n'est nullement une nouvelle version d'un produit existant en matière d'E.A.O. ('*Enseignement assisté par ordinateur*').

Son originalité réside dans le fait qu'il propose au professeur d'introduire les exercices de son choix, de créer un dictionnaire reprenant des termes propres aux mathématiques ainsi que des fiches d'aide pour les élèves, etc. Il permet également à l'élève de se corriger sans pour autant lui donner directement la bonne réponse, de l'aider tout au long de la mise en équation et de la résolution du problème en lui offrant différents outils, etc.

Nous avons essayé d'élaborer un didacticiel qui ne soit pas trop rigide. Nous voulions que les élèves réfléchissent au lieu d'agir mécaniquement pour la mise en équation et la résolution d'un problème. Par exemple, l'élève peut commencer l'exercice par l'étape de mise en équation du problème. Pour autant que toutes les étapes demandées par le professeur soient complétées, l'élève conserve une certaine marge de liberté pour résoudre le problème.

Les motivations

A travers ce mémoire, nous avons voulu concilier les différents concepts acquis durant notre cycle d'études avec nos connaissances pédagogiques assimilées lors de notre agrégation.

Nous sommes parfois loin d'imaginer la complexité qu'entraîne le développement d'un tel projet. En effet, tout paraît simple au départ mais, bien vite, nous nous apercevons que la tâche du professeur, pour faire passer certaines notions, n'est pas toujours une chose aisée ... et encore moins pour le concepteur informatique. Ce dernier doit décomposer chaque étape principale en une multitude de procédures. La réalisation de ce cas concret d'analyse d'un problème, consacré à l'enseignement, est un véritable défi. Nous nous permettons même de dire qu'il s'agit d'un dépassement de soi.

Nous sommes seuls face à un projet et notre tâche est d'avancer le plus loin possible pour lui donner une certaine « viabilité ». Nous devons tout concevoir, tout est à faire, en partant de la constitution d'un cahier des charges pour terminer sur l'implémentation et l'exécution du programme.

La dernière raison du choix de ce mémoire est de voir aboutir le fruit de nos recherches dans l'exécution d'un didacticiel, éveillant, nous l'espérons, de vifs encouragements quant à une prolongation de notre travail.

Le cheminement du travail

Dans un premier temps, nous avons tenu à faire plus amples connaissances avec quelques-uns de nos utilisateurs potentiels. Nous pensons qu'il est capital de comprendre l'environnement, les attentes et les craintes des élèves et des professeurs avant de se lancer proprement dit dans l'élaboration de notre didacticiel. Il est certain que l'introduction de l'ordinateur, comme support d'enseignement, pose de nombreux problèmes. Il est important d'en relever quelques-uns; ce qui nous a permis de percevoir quelques difficultés, tantôt des élèves, tantôt des professeurs, à se servir de l'outil informatique et d'y remédier. Comment insérer sur une disquette le savoir, la pédagogie, la méthodologie et les mille et une façons d'enseigner de plusieurs professeurs pour une matière ? Soyons réalistes, ce n'est pas possible! Dans le premier chapitre, nous présenterons ces différents aspects.

Les objectifs pédagogiques poursuivis, l'analyse mathématique du projet qui permettra de le découper en deux grands modules (le module du professeur et le module de l'élève) ainsi que la présentation de divers outils d'aide, feront l'objet de notre second chapitre.

Dans le troisième chapitre, nous exposerons les différents concepts et les diverses stratégies que nous avons développés pour analyser syntaxiquement et sémantiquement ainsi que pour manipuler des expressions algébriques.

Dans le quatrième chapitre, nous nous attaquerons à une des grandes tâches qui nous incombent : l'interface. Nous verrons qu'elle doit être la plus élaborée possible pour rendre plus conviviale l'interaction avec l'utilisateur, que ce soit le professeur ou les élèves, et pour l'aider dans l'accomplissement de sa tâche.

Le dernier chapitre sera consacré à l'analyse logicielle. Nous détaillerons pour les procédures et les fonctions les plus complexes, l'analyse fonctionnelle, l'analyse conceptuelle et l'implémentation.

Nous conclurons notre travail en faisant un petit bilan sur ce qui a été réalisé.

Le choix du langage

Ci-dessus, nous avons déjà insisté sur l'importance d'une interface conviviale. Le langage de programmation choisi devra donc offrir des facilités graphiques pour représenter des composants visuels.

De plus, il ressort de notre analyse pédagogique que les utilisateurs vont probablement effectuer de nombreuses manipulations (souris, menus, ...). Nous aurons donc besoin d'un grand nombre d'éléments visuels interactifs. D'un point de vue conceptuel, ces éléments sont en fait des objets définis par un certain nombre de propriétés et d'événements. Nous avons besoin d'un langage orienté objet.

Il nous faut aussi pouvoir conserver des données de façon permanente. Un outil offrant la gestion de bases de données serait non négligeable.

Delphi est le dernier outil de développement sous Windows de Borland. Ce produit combine des outils visuels et une technologie d'accès évolutif aux bases de données. A la différence des autres outils existants sur le marché, il permet de développer rapidement des applications Windows et client/serveur. Son architecture présente les avantages suivants :

- Développement d'applications très performantes
- Possibilité de réutiliser le code
- Développement rapide d'applications
- Evolutivité de l'accès aux bases de données.

En outre, Delphi utilise un langage structuré orienté objet, le Pascal Objet. Il supporte donc les concepts de programmation avancés tels que l'encapsulation, l'héritage et le polymorphisme. Pour ces différentes raisons, nous avons choisi cet outil pour l'implémentation de notre didacticiel.

Chapitre 1

Vers un enseignement "adaptatif", personnalisé

Introduction

Derrière un ordinateur, se cache un être humain. Ce premier chapitre attirera notre attention sur le fait, qu'en tant qu'informaticien, nous ne pouvons pas entrer dans une classe, installer notre didacticiel et considérer que le reste se fera sans problème. Nous devons tenir compte de certains principes pédagogiques et méthodologiques lorsque nous nous lançons dans la réalisation d'un didacticiel.

Dans un premier temps, nous définirons les termes suivants : enseignement adaptatif, personnalisé, psychologie cognitive et dimension affective. Pour chacune de ces définitions, nous y avons attaché un concept, une idée en rapport avec la conception du didacticiel.

La mise en place d'un enseignement adaptatif demande de connaître un certain nombre d'informations concernant la structure cognitive et/ou affective qui caractérise l'élève. Nous analyserons quelques-unes de ces informations tout en les mettant en relation avec notre travail : la maîtrise des prérequis et des objectifs assignés aux cours, les représentations préexistantes et les variables affectives (la motivation scolaire, les appréhensions des élèves).

L'avènement des nouvelles technologies de l'information dans l'enseignement peut être vu sous plusieurs angles. Nous vous proposerons quelques conceptions de l'outil informatique dans un système éducatif. Mais, l'intégration de ces nouvelles technologies crée une série de déséquilibres dont la résorption dépend à la fois de la résistance au changement et des mesures d'accompagnement prises pour les adopter. Nous citerons quelques facteurs qui exercent une influence significative sur leur rejet ou leur acceptation.

1.1 Quelques définitions et concepts

Nous souhaitons clarifier ce que nous appelons *enseignement adaptatif, personnalisé, psychologie cognitive* et *dimension affective*.

1.1.1 Un enseignement adaptatif, personnalisé

Selon [DEPO87], un système d'enseignement adaptatif se caractérise par la présence d'une relation bidirectionnelle entre l'organe qui apprend et celui qui enseigne. La relation OE-OA représente le transfert d'information du système d'enseignement vers l'apprenant. La relation OA-OE décrit l'existence d'un retour d'information de l'apprenant vers l'enseignant.

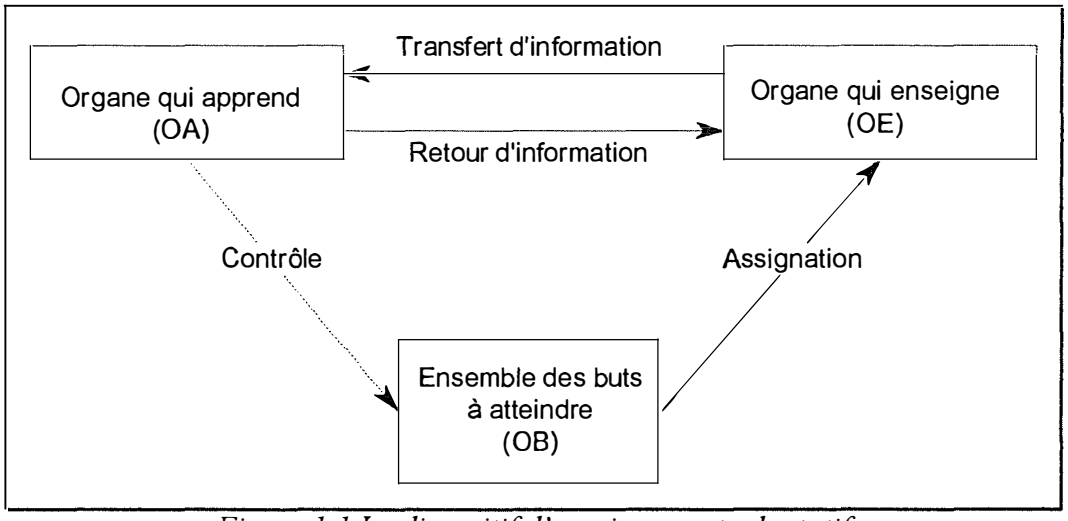


Figure 1.1 Le dispositif d'enseignement adaptatif (Modèle comportementaliste)

L'enseignant exploite l'information reçue afin de réguler son action de manière à atteindre les buts qui lui sont assignés (relation OB-OE). La relation OA-OB désigne la possibilité offerte à l'élève d'exercer un contrôle sur les buts qui seront poursuivis.

DEPOVER définit l'enseignement adaptatif en ces termes : *« Nous parlerons d'enseignement adaptatif uniquement lorsque les possibilités de régulation existent durant l'ensemble du processus d'enseignement, autorisant ainsi un ajustement immédiat des actions d'enseignement au comportement d'apprentissage manifesté par l'apprenant » [DEPO87].*

Le concept d'enseignement *personnalisé* rejoint la définition de l'enseignement adaptatif. Il s'agit d'adapter des situations d'apprentissage aux caractéristiques individuelles de chaque élève.

La psychologie comportementaliste est une des disciplines qui s'est intéressée aux facteurs externes à l'individu. L'enseignement assisté par ordinateur a bâti ses fondements dans ce courant.

Pour un comportementaliste, l'apprentissage, concept central de cette théorie, est une modification du comportement consécutive à un conditionnement. *« L'élève ne peut aborder une étape du programme d'apprentissage que si l'étape précédente est parfaitement maîtrisée [...]. L'enseignant laisse à la machine la transmission d'un savoir formel et se focalise préférentiellement sur la réflexion relative à ce savoir et l'enseignement de domaines plus 'subjectifs', la définition des objectifs à atteindre, etc. Il peut également consacrer plus de temps aux élèves en difficulté » [LECO93].* Notre didacticiel a été conçu pour accomplir des tâches fastidieuses telles que la résolution de l'équation du problème pour l'enseignant (cfr. 2.2.3.1 Le module du professeur, page 29), ce qui permet à ce dernier de se concentrer sur d'autres objectifs. Le didacticiel se caractérise par une facilité d'utilisation car aucune connaissance en informatique n'est requise pour créer ou résoudre des exercices. Le professeur peut se concentrer sur les aspects pédagogiques du système d'apprentissage, d'organisation, d'évaluation et de communication des connaissances. Nous insistons sur le fait que cet outil informatique doit être vu comme un complément et non en tant que remplacement du professeur. La présence de l'enseignant dans la classe est importante car elle permet de remédier à une carence du modèle comportementaliste, à savoir, la non prise en considération des représentations existantes de l'élève, de la motivation, de la dimension affective, etc. Nous reviendrons sur l'importance des contacts humains et des caractéristiques affectives. Malheureusement, nous ne pouvons pas démentir que l'introduction des technologies de l'information dans notre société est une des causes du chômage dans de nombreux pays.

1.1.2 La psychologie cognitive

Le cognitivisme, contrairement au modèle comportementaliste, s'est attaché aux facteurs d'apprentissage internes de l'individu. *« Selon ce courant, l'enseignant doit tenir compte du savoir antérieur de l'individu afin de le confronter judicieusement avec des informations nouvelles en vue d'un changement de conceptions. Un domaine de recherche propre concerne la métacognition, qui décrit la connaissance qu'une personne a de ses propres processus cognitifs, de leurs produits et de la procédure qu'elle emploie pour les réguler. La finalité pratique de ces recherches est de permettre à l'individu d'apprendre à apprendre » [LECO93].* Ce savoir antérieur diffère d'un individu à l'autre. L'arrivée des multimédias dans notre société tels que la télévision, la radio, l'audiovisuel ... a contribué à l'acquisition de nouvelles connaissances (bonnes ou mauvaises) en dehors du cadre scolaire. Aussi, un élève, qui a la possibilité d'utiliser un ordinateur chez lui, aura déjà une opinion (bonne ou mauvaise) de l'ordinateur par rapport à un autre élève qui n'en possède pas.

« Tous les individus n'ont pas les mêmes capacités intellectuelles et surtout n'utilisent pas les mêmes procédures de résolution de problèmes » [LECO93]. Il faudrait pouvoir adapter notre didacticiel au niveau de connaissance de l'élève. Mais, nous devons reconnaître les limites de notre didacticiel. Seul l'enseignant peut déterminer s'il est plausible qu'un élève ait trouvé la solution d'un problème en quelques étapes. Par ailleurs, en admettant qu'il soit possible au professeur de signaler à l'ordinateur quels sont les bons éléments de sa classe, nous devrions concevoir un didacticiel qui offre une pédagogie et un contenu complètement différents.

Un élève dit « doué » pourrait bénéficier d'une plus grande marge de liberté dans son apprentissage. En effet, il pourrait se débrouiller seul pour une partie de la matière. Par exemple, si les prérequis (codage littéral et numérique, etc.) ne sont plus à remettre en cause et qu'il sait résoudre une équation du premier degré à une inconnue, il peut d'emblée se lancer dans la résolution d'un problème du premier degré à une inconnue. Une aide est fournie dans ce système et l'élève peut la consulter à tout moment.

A l'opposé, un élève dit « faible », se demandant encore, par exemple, si le triple du double de quatre, est la même chose que le double du triple de quatre, aurait besoin d'une bonne assistance pour la mise en équation du problème. Les explications fournies devront être les plus claires possibles. Nous devrons accroître le nombre d'étapes qui conduisent de la lecture de l'énoncé à la résolution du problème.

1.1.3 La dimension affective

L'homme attribue à certaines de ses conduites une signification affective, émotionnelle qui vient, en partie, masquer le caractère purement instinctif de la conduite. C'est ce que nous appelons la *dimension affective*.

Un des nombreux souhaits de certains adolescents est d'être reconnus comme « une personne, un être humain » et non pas comme « des moins que rien, des individus sans personnalité, des numéros ». Les adolescents sont perpétuellement à la découverte d'eux-mêmes. Un des rôles du professeur consiste à extraire le meilleur de ses élèves afin qu'ils acquièrent une bonne image d'eux-mêmes, assurant dès lors de meilleurs résultats au niveau de leur travail. N'oublions pas qu'entre onze et seize ans, *« l'image propre (celle que l'on se forme de soi-même) et l'image dite 'sociale' (l'opinion de soi qu'on attribue aux autres, parents, camarades et professeurs) tendent à s'uniformiser » [JOUR94]. « L'échec ou la réussite scolaire sont largement dus à la manière dont l'élève perçoit ou non qu'il a compris ou non » [LECO93].*

Notre précurseur, B.S. BLOOM[1], ne nous démentira pas. En effet, il avait établi que les caractéristiques affectives de départ, à savoir l'intérêt de l'élève pour la matière, son attitude face à la tâche et l'image qu'il se fait de lui-même en tant qu'apprenant, jouent un rôle prépondérant dans la réussite d'un apprentissage. Aussi, tout enseignant doit être conscient qu'il participe à l'histoire pédagogique de chacun de ses élèves et doit tenir compte de la manière dont il a perçu son expérience scolaire antérieure. L'adolescence est une période de conduites de risques et d'oppositions qui se traduisent par un comportement antisocial et pas seulement dans le cadre scolaire. Se construire une identité ne se fait jamais sans souffrance. L'adolescent mal à l'aise, voire agressif avec son entourage, va se réfugier dans son propre univers. L'une des tâches de l'enseignant sera de prendre en considération le vécu subjectif de chacun de ses élèves, ainsi que ses caractéristiques affectives. Encore faut-il qu'il en ait le temps et les moyens?

Une fois encore, notre didacticiel doit s'avouer vaincu d'une certaine manière. Mais, est-ce vraiment une défaite? Certes, l'ordinateur est dépourvu de sentiments humains. En tant que concepteur, nous pouvons lui donner une certaine forme de vie en soignant notre interface. Ce que l'élève apprendra à travers notre didacticiel est étroitement lié à la manière dont la situation d'apprentissage lui sera présentée.

[1] Notes de synthèse de J. Beckers, [DONN95].

En particulier, nous devrons attirer son attention sur les éléments les plus significatifs tout en veillant à déclencher chez lui les démarches mentales qui lui permettront d'acquérir les savoir-faire impliqués par les objectifs du cours. Les moyens dont nous disposons sont fort variés : envois de messages réconfortants lors des échanges interactifs telles que *« A toi de jouer ! », « Tu es sur la bonne voie, continue !»*, couleurs, dessins, animations, effets sonores. Le problème essentiel ne réside pas dans la disponibilité des moyens, mais plutôt dans leur utilisation judicieuse par rapport à la définition des finalités éducatives fixées. Nous en reparlerons au chapitre 4 consacré à l'interface.

De plus, le fait que l'ordinateur ne possède pas cette dimension affective peut avoir certains avantages. Par exemple, restant objectif, le didacticiel sera impartial dans la répartition des points pour un exercice (une grille d'évaluation étant préétablie). Bien entendu, le professeur peut toujours descendre ou rehausser la cote, indépendamment du résultat fourni par l'ordinateur. L'élève se retrouvant seul devant la machine, le phénomène de tricherie peut également être écarté du moins si notre didacticiel prévoit de protéger l'accès aux exercices résolus et si les machines sont suffisamment espacées les unes des autres.

1.2 L'acquisition d'informations concernant la structure cognitive et/ou affective qui caractérisent l'élève

Si nous réfléchissons aux valeurs essentielles du travail d'un enseignant, nous remarquons que son travail doit être adapté à la classe, à l'option de l'élève. Une solution remarquable serait un enseignement adaptatif, personnalisé. Pour ce faire, nous devons disposer d'un certain nombre d'informations concernant la structure cognitive et/ou affective caractérisant l'élève. Quelles sont-elles?

1.2.1 La maîtrise des prérequis

L'enseignant est chargé de modifier et d'enrichir les structures cognitives de l'élève. Mais, il doit être prudent. En effet, s'il souhaite qu'un changement se produise au niveau de ces structures, il doit s'assurer que ses apprenants ont maîtrisé les prérequis, les connaissances et les aptitudes de départ nécessaires à l'apprentissage. J.B. CARROLL et B.S. BLOOM[2], partisans d'une pédagogie centrée sur la maîtrise des compétences, montraient que la maîtrise des prérequis conditionnait, dans une large mesure, le succès d'acquisition de nouvelles connaissances.

Il ne suffit pas que l'élève ait les prérequis nécessaires pour aborder un apprentissage, il faut aussi que ceux-ci soient facilement disponibles lorsque l'élève en a besoin (informations verbales). Il est donc important de construire un réseau sémantique contenant de nombreuses connections afin de faciliter leur réactivation dans la mémoire de travail au moment souhaité.

Le professeur pourrait avoir recours à une évaluation sommative, ce qui lui permettrait de faire le point sur une somme de connaissances antérieures, plus précisément, sur la compréhension des prérequis. Nous pourrions imaginer, tout en poursuivant notre objectif d'instaurer un enseignement adaptatif, que l'évaluation soit, elle aussi, adaptative (évaluation continue) comme le proposent MONTOISY et DEPOVER[3]. Il s'agit de soumettre à l'élève des questions relatives aux prérequis spécifiques en tenant compte de la qualité de ses réponses aux questions qui lui ont été précédemment proposées.

Dans le cadre de ce mémoire, nous avons émis l'hypothèse que le didacticiel pourrait prévoir quelques exercices en guise d'introduction. Ceux-ci permettraient d'une part, de savoir si les prérequis ont été suffisamment maîtrisés et, d'autre part, de se familiariser avec l'environnement de travail (cfr. 2.2.1 Les prérequis, page 27).

[2] Notes de synthèse de J. Beckers, [DONN95].

[3] MONTOISY et DEPOVER, *'Analyse de cinq stratégies d'évaluation adaptative gérées par ordinateur'*, 1987, [DONN95].

1.2.2 La maîtrise des objectifs assignés aux cours

Désireux d'améliorer la qualité de l'enseignement, B.S. BLOOM proposait d'organiser la tâche d'apprentissage, c'est-à-dire de déterminer avec précision les objectifs à atteindre. Dans la perspective d'un enseignement adaptatif, il serait intéressant de suggérer à l'élève un itinéraire personnalisé, lié à son niveau d'apprentissage. En effet, l'élève pourrait apprendre à son rythme les différentes étapes de cet itinéraire (unités d'apprentissage). Par exemple, si l'élève estime disposer de certaines compétences que le cours se propose de lui faire acquérir, l'enseignant évaluera ses connaissances. En fonction des résultats, l'enseignant déterminera si l'élève peut poursuivre son apprentissage ou bien, s'il est préférable qu'il reprenne l'apprentissage de l'objectif testé (prise de décision de remédiation).

Par rapport à notre didacticiel, les grandes étapes de résolution d'un problème du premier degré à une inconnue sont regroupées dans la fenêtre principale. L'élève reste libre de commencer là où il le désire. L'itinéraire personnalisé, lié à son niveau d'apprentissage serait plutôt tracé par la résolution d'exercices de plus en plus complexes.

1.2.3 Les représentations préexistantes

Nous appellerons « représentations préexistantes » les connaissances appartenant au répertoire cognitif de l'élève qui ne sont pas le résultat d'une action d'enseignement systématique. Elles constitueront des structures de connaissances à partir desquelles l'action pédagogique pourra se développer. Dans d'autres cas, il sera utile de les identifier afin de persuader l'élève de leur inadéquation.

[AUSU68] considère que certaines représentations préexistantes pourraient servir de point d'ancrage à l'acquisition de nouvelles connaissances. Il est vrai que l'apport des technologies de l'information est considérable pour l'enseignement. Nous pensons notamment à la télévision qui diffuse des programmes culturels, le journal donnant des nouvelles en provenance du monde entier, etc. Mais, il y a le revers de la médaille. Si l'émission « Questions pour un champion » entretient notre culture générale, il n'en est pas de même des films d'horreur ou de science-fiction. Le fait que « Superman » vole n'est absolument pas lié à un quelconque principe physique (terrien).

Selon [TARD92], *« on ne peut apprendre que ce qu'on connaît déjà »*. L'élève doit avoir dans sa mémoire à long terme des connaissances qui lui permettent d'établir des liens avec les nouvelles informations qui lui sont fournies, sinon il ne peut les traiter de façon significative. De plus, l'élève ne peut éliminer de sa mémoire à long terme des connaissances pour les remplacer par d'autres présentées par l'enseignant. Cependant, le processus d'apprentissage est cumulatif, c'est-à-dire que les nouvelles connaissances s'associent aux anciennes soit pour les confirmer, soit pour y ajouter de l'information, soit pour les nier (négociation - action pédagogique).

1.2.4 Les variables affectives

« Apprendre se fait rarement sans effort » [DONN95]. Nous l'avons vu précédemment par le biais de la théorie de B.S. BLOOM. Il ne faut pas perdre de vue que les caractéristiques affectives laissent dans notre mémoire des empreintes tantôt positives, tantôt négatives. Citons, entre autres, la motivation scolaire et les appréhensions des élèves.

1.2.4.1 La motivation scolaire

« Pour qu'un enseignement soit efficace, il est nécessaire de maintenir la motivation spécifique à un niveau élevé par la présentation de situations adaptées aux intérêts des apprenants. Dans cette perspective, LUTZ a montré expérimentalement que le recours à des animations, afin de présenter des feed-back plus attrayants, permettait, chez de jeunes enfants, d'améliorer la motivation et l'intérêt pour la tâche dans un cours par ordinateur » [DEPO87]. Le didacticiel devrait prévoir des modules permettant d'insérer des effets sonores, des animations. Une fois encore, nous voyons l'importance de concevoir une bonne interface. Cependant, ce souci d'améliorer la motivation de l'élève ne doit toutefois pas prendre le pas sur les aspects formatifs de la situation d'apprentissage. L'exemple suivant ne peut pas être plus explicite : *« Nous avons eu l'occasion d'examiner un logiciel d'arithmétique dans lequel chaque réponse correcte de l'élève lui permettait d'accéder à un jeu d'arcade. Non seulement le temps passé à jouer était prohibitif par rapport à celui consacré à l'apprentissage mais, en plus, la fin de la session d'apprentissage était contrôlée, non pas par des critères pédagogiques, mais par le succès du jeu » [DEPO87].*

1.2.4.2 Les appréhensions des élèves

Dans un contexte d'apprentissage, le professeur se heurte aux inquiétudes des élèves : « Ah, les maths! J'ai toujours été nul en maths, cela ne changera jamais ». Il est certain que le professeur devra « réconcilier » l'élève avec sa matière, avant d'entamer un quelconque apprentissage. Aussi, le professeur devra user d'habilité pour convaincre l'élève que, lui aussi, peut faire des mathématiques, qu'il ne s'agit pas là d'une incapacité innée, mais d'un manque de confiance en lui, d'une incompréhension. Une fois encore, le rôle qu'assumera le professeur est fort important.

Le didacticiel a lui aussi un rôle important. Lors des échanges entre l'ordinateur et l'élève, il faut que chaque message soit le plus significatif et encourageant possible. Les feed-back doivent contribuer à soutenir la motivation de l'élève mais également à répondre par des encouragements, des messages personnalisés. Selon [DEPO87], « *BETZ a montré que les élèves commettant un nombre important d'erreurs faisaient preuve d'un niveau de motivation inférieur et d'appréhension plus élevé lorsque le feed-back attirait avec trop d'insistance l'attention sur leurs faiblesses* ».

De plus, comme le dit le proverbe turc *« une personne sans problèmes, dans la tombe »*, signifiant que tout le temps que nous vivrons, nous rencontrerons des problèmes. Si nous arrivons à résoudre un problème, un autre se crée. Il se peut que nous puissions contribuer à régler certaines difficultés scolaires mais d'autres problèmes seront générés. Les élèves tout comme leurs professeurs devront s'adapter à ce nouveau mode d'enseignement. L'ordinateur n'est pas une « bombe ». Il ne va pas exploser si un utilisateur commet une erreur de manipulation.

Nous avons vu que la maîtrise des prérequis et des objectifs assignés aux cours pouvait s'acquérir et être évaluée de manière individuelle. Cependant, il faut tenir compte des atouts et/ou des inconvénients des informations telles que le rôle prépondérant des représentations préexistantes, les variables affectives. Selon [DEPO87], la pertinence de ces variables est déterminante lorsqu'il s'agit d'adapter le processus d'enseignement aux particularités individuelles. Il cite également, les aptitudes intellectuelles (l'efficacité d'une méthode d'enseignement dépend largement des sujets avec lesquels elle est utilisée), les stratégies de traitement de l'information, les variables liées au temps (le temps est une variable pédagogique déterminante pour les auteurs de la pédagogie de la maîtrise - J.B. CARROLL), etc.

1.3 L'apport des nouvelles technologies de l'information et les problèmes d'insertion

Il est clair qu'il est impossible pour un professeur de préparer autant de cours qu'il y a de niveaux différents de connaissances au sein de sa classe. Il peut prévoir des leçons de rattrapage où il invite les élèves plus faibles à améliorer leur niveau de connaissance. Il peut également donner des exercices supplémentaires aux élèves les plus assidus de sa classe pendant qu'il réexplique une partie de sa leçon aux autres apprenants. Néanmoins, il n'arrivera pas à se mettre au niveau de chaque élève. Le souci de particulariser l'enseignement en fonction des savoir-faire de chacun est loin d'être une idée nouvelle. Souvenons-nous des aventures de Pantagruel et de Gargantua ...

1.3.1 Quelques conceptions de l'outil informatique

L'ordinateur peut être employé de différentes façons, en voici quelques-unes.

Tout d'abord, le professeur peut apprendre à ses élèves à ***utiliser l'outil informatique***. Dans cette optique, l'informatique est définie comme une nouvelle matière à enseigner, faisant partie du programme des cours. Nous parlons alors de cours d'informatique, de bureautique, etc. L'idéal serait que chaque élève dispose de son propre ordinateur, de la même manière qu'il dispose d'un bic et d'un crayon dans son cartable. Nous supposons qu'une classe spécialement équipée de matériel informatique soit mise à la disposition du professeur titulaire de ce cours, telle le local de géographie avec ses cartes du monde affichées sur les murs.

Ensuite, l'outil informatique peut servir comme un ***support de cours.*** Il est utilisé pour remplacer notre traditionnel tableau noir, les cartes géographiques, etc. Nous pouvons imaginer un écran géant alloué par classe sur lequel le professeur ferait défiler sa matière au lieu de la présenter au tableau. Chaque classe disposerait de son outil informatique.

Le dernier usage envisagé et qui nous intéresse particulièrement dans le cadre de ce mémoire, c'est de considérer l'ordinateur comme un livre « évolué ». Il s'agirait d'un ***support pour les exercices et pour la théorie***. Dans chaque classe, il faudrait aménager plusieurs ordinateurs, pas nécessairement un par élève. Selon la taille de l'écran, nous pourrions dire qu'un ordinateur pour trois élèves serait suffisant.

1.3.2 Quelques facteurs à prendre en considération

Nous évoluons vers un monde informatisé, il est indispensable que l'école se mette au goût du jour. Pour cette raison, de grands projets sont en train de se développer un peu partout dans le monde, poussant l'institution scolaire à utiliser les technologies de l'information.

Cependant, en nous basant sur les ouvrages de D. WATSON - D. TINSLEY[4] et de Ch. DEPOVER[5], la généralisation au niveau de l'intégration de ces technologies est un grave problème. En effet, il semblerait que les technologies de l'information ne puissent s'incorporer que dans un cadre spécial où d'importants investissements (personnel et matériel) peuvent y être fournis.

1.3.2.1 L'importance de l'emplacement du matériel

Le livre de D. WATSON et D. TINSLEY suggère que les nouvelles technologies de l'information ne doivent pas être regroupées dans une pièce spéciale mais présentes dans chaque classe (rétro-projecteur, vidéo, ordinateur relié éventuellement à un LAN[6], etc.). Ainsi, elles viendraient s'incorporer dans le décor comme leurs confrères le tableau et la craie. Il est vrai que l'endroit où le matériel sera mis à la disposition des élèves, est important.

En effet, envisageons un professeur de mathématiques qui souhaite faire une étude de fonction à l'aide d'un logiciel. S'il doit aller dans une autre pièce que celle où il a l'habitude d'enseigner, les élèves, changeant de cadre, peuvent être perturbés (distraits, bavards, peu attentifs, ... pensant que cela ne fait pas partie du cours).

1.3.2.2 L'aspect financier

Le coût reste le principal obstacle à la diffusion sur une large échelle de l'ordinateur en tant qu'outil de formation. Chaque école dispose d'un budget qu'elle alloue pour l'installation du matériel technologique - informatique. En fonction de ce montant, les écoles sont plus ou moins bien équipées. Mais, débloquer des fonds pour quelque chose qui évolue rapidement dans le temps, requiert certaines prises de décision au préalable, surtout en période de restructuration de l'enseignement.

Premièrement, « *Comment choisir un micro?* » sachant que d'ici 18 mois, il faudra peut-être racheter un nouvel ordinateur et/ou logiciel. Windows 3.11 a fait place à Windows 95 et les revues spécialisées annoncent déjà que la prochaine version de Windows est en phase de test.

Deuxièmement, l'école veillera à acheter du matériel de bonne qualité tel que des écrans 'Low radiation' afin de ne pas détériorer la santé de ses professeurs et de ses élèves. Finalement, il faut veiller à ce qu'il y ait un nombre suffisant de machines dans la classe, afin de ramener le ratio élève/ordinateur le plus bas possible. Cependant, qui dit acquisition de nouveaux logiciels, dit aussi apprentissage de ceux-ci. De nouveau, il faut tenir compte d'un certain temps d'adaptation de la part des professeurs (et des élèves). Mais, l'école sera-t-elle toujours apte à subvenir à la formation de ses enseignants? De plus, il ne faudrait pas que les professeurs passent la moitié de leur temps à apprendre le fonctionnement de l'ordinateur et l'autre moitié, à adapter le sujet de leur leçon en fonction des didacticiels offerts sur le marché.

---

[4] [WATS95]

[5] [DEPO87]

[6] Local Area Network.

Par ailleurs, avant de se lancer dans l'achat de tout ce matériel, ne serait-il pas plus intéressant, moins coûteux d'engager un autre enseignant à la place? La qualité de l'enseignement s'en verrait, certes, améliorée mais les élèves ainsi que leurs professeurs seraient bien loin du monde réel. Il faut mesurer l'aspect financier mais ne pas négliger les conséquences de ce choix. Nous allons envisager les différentes composantes qui contribuent au coût financier d'une application d'enseignement par ordinateur, en nous basant sur les écrits de DEPOVER. Il s'agit d'une configuration d'ordinateur en site propre auquel sont raccordés plusieurs centaines de terminaux.

LES COMPOSANTES	LA REPARTITION DES COUTS
Le coût du hardware (unités centrales, périphériques, etc.) et les terminaux.	CPU : 15 % Terminaux : 45 %
Le coût des télécommunications, variable selon qu'il s'agisse d'un réseau local ou d'un réseau à longue distance, selon la vitesse et la fiabilité assurées au transfert des informations	Communications (locales uniquement) : 15 %
Le coût du logiciel de base, qui comprend le coût du système d'exploitation, des langages, de certains utilitaires.	Coût du software (droit d'usage) : 5 %
Le coût d'achat, de location, de développement des didacticiels. Le coût de maintenance (estimé habituellement en pourcentage du prix d'achat du matériel)	Maintenance : 8 %
Les coûts en personnel de contrôle, de développement, d'encadrement pédagogique	Frais de personnel : 12 %

Tableau 1.1 Ventilation des coûts d'un système E.A.O.

Nous voyons que le facteur de coût le plus important est associé au prix des terminaux suivi par ceux liés à l'usage du CPU et des lignes de communication. Si nous passons d'une configuration en site propre à une configuration décentralisée, les coûts relatifs aux télécommunications peuvent monter jusqu'à plus de 30 %. La part très faible du coût total consacré aux didacticiels correspond, dans ce type de configuration, à l'hypothèse, certes peu réaliste, où l'ensemble du software nécessaire à la mise en oeuvre d'une application est déjà disponible.

1.3.2.3 L'acceptabilité de l'ordinateur en tant qu'outil d'enseignement.

L'intégration de l'outil informatique fait face à de nombreuses résistances. *« On imagine aisément la distance à parcourir, par un enseignant habitué aux structures rigides qui caractérisent la plupart des systèmes scolaires nationaux, pour arriver à une exploitation efficace d'un outil d'enseignement adaptatif » [DEPO87].* Nous avons retenu quelques facteurs énumérés par DEPOVER qui jouent un rôle déterminant sur le rejet ou l'adoption d'une nouvelle technologie par le milieu éducatif.

A. Le contrôle par l'enseignant

Selon DEPOVER, *« une innovation technologique qui donne aux enseignants l'impression de les déposséder d'une partie de leurs prérogatives éducatives a peu de chances d'être adoptée » [DEPO87].* Nous pouvons observer deux grandes tendances au sein des enseignants. Certains sont favorables mais ils insistent pour conserver leurs prérogatives en matière de choix des objectifs, d'évaluation de leur enseignement; d'autres, par contre, sont carrément hostiles à l'implantation de l'informatique dans leur école. Un climat d'extrême méfiance peut exister à l'égard d'une telle innovation. Nous pouvons facilement imaginer que cela ne se réalisera pas sans problème puisqu'aujourd'hui, un faible pourcentage d'enseignants utilisent les technologies de l'information.
En effet, les professeurs mis en place en ce moment, habitués à leur méthode de travail, devront se familiariser à un enseignement d'une nouvelle génération. Une préparation « technique » des enseignants pourrait leur permettre de lever leurs craintes liées au maniement de l'outil informatique. Cette préparation à l'introduction des ordinateurs pourrait dans l'enseignement être complétée par une acquisition de nouvelles méthodes d'organisation du travail scolaire qui mettrait davantage l'accent sur l'individualisation et l'initiative de l'élève.

Suite à la lecture du rapport de recherche [BRUL93], nous constatons que l'exploitation de nouveaux outils dans les classes entraînent d'éventuels changements au niveau des rôles de l'enseignant. Et, par la même occasion, le développement de nouveaux besoins ou compétences sont requis pour assumer ce nouveau style d'enseignement. Les représentations de l'enseignant à propos de l'outil informatique pourraient provoquer des difficultés pour utiliser cet outil. Or, *« La représentation qu'a l'enseignant de son rôle et la manière dont il le met en oeuvre découlent de ses conceptions de l'enseignement, elles-mêmes influencées par ses conceptions de l'apprentissage de l'élève » [BRUL93].*

Par conséquent, si les perceptions et/ou circonstances d'apprentissage sont modifiées, le rôle pourrait aussi s'en trouver changé dans le sens où le professeur ne serait plus un transmetteur de savoir. L'élève le perçoit alors comme un « facilitateur », une « personne ressource » (conseiller) ou encore comme un gestionnaire d'environnements d'apprentissage.

Nous parlons alors d'un style d'enseignement *associatif*. Le rôle essentiel de l'enseignant est de faciliter les apprentissages individuels et collectifs. Selon F.Torchon, *« l'élève n'est pas un récepteur passif, mais il assimile par lui-même son programme et s'exerce personnellement. L'enseignant contrôle le travail et le guide librement. Les élèves se forment par le travail personnel ou celui de groupe, en agissant. L'enfant participe activement à l'élaboration même des connaissances qui mettent en jeu son initiative et sa créativité » [BRUL93].*

Le didacticiel offre deux facettes dont une est dédiée à la préparation des cours du professeur. Ainsi, il peut choisir les exercices proposés aux élèves, du plus simple au plus compliqué. De plus, nous avons tenu à faire participer quelques professeurs de mathématique à ce travail. Prenant en considération leurs remarques, leurs suggestions, nous avons pu mener à bien ce projet.

B. La souplesse d'utilisation - la fiabilité et la simplicité d'utilisation

Un nouveau média a d'autant plus de chance d'être accepté que son utilisation est peu contraignante pour son utilisateur (élève ou professeur). En ce qui concerne l'élève, nous avons essayé de lui offrir un ensemble de procédures permettant d'une part, de confirmer la réponse correcte qu'il propose; d'autre part, de l'aider à découvrir la réponse correcte par l'intervention de différents outils d'aide (cfr. 2.3 Les outils d'aide, page 37).

Les enseignants sont généralement réticents à employer un outil qui risque de leur poser des difficultés d'utilisation. Lorsqu'un professeur se sert de notre didacticiel, il est libéré de toute connaissance technique en informatique. Le logiciel est simple à utiliser et le manuel d'utilisation (cfr. Annexe 2) lui fournit les renseignements nécessaires à son bon fonctionnement. En cours d'exécution du programme, l'utilisateur peut faire appel à tout moment à l'aide prévue dans ce système. Le professeur est donc dégagé de toutes contraintes liées au fonctionnement de l'ordinateur dans la mesure où il est habitué à travailler avec un environnement multi-fenêtré tel que Windows 3.x. Un environnement technique peut néanmoins interférer avec le corps professoral dans certaines limites. Chacun a un rôle distinct. Le professeur verra l'outil informatique comme un support à ses cours ou comme une matière à enseigner. Le technicien s'occupera de l'installation et de la maintenance du matériel.

C. Un rapport coût-efficacité positif

Selon DOYLE et PONDER[7], de nombreuses innovations ont échoué parce qu'elles exigent de l'enseignant des investissements en temps et en effort trop importants par rapport aux profits qu'ils pourraient espérer en tirer dans son enseignement. Dans certaines écoles, les formations sont laissées aux frais des professeurs et bien souvent en dehors de leur horaire. La disponibilité et l'accessibilité au matériel laissent à désirer. Un enseignant est parfois même obligé de se procurer un ordinateur pour travailler chez lui.

L'apport des technologies de l'information dans l'enseignement n'est pourtant pas à négliger. Certains outils informatiques constituent un véritable gain de temps pour les tâches fastidieuses et routinières. Par exemple, la rapidité de la technologie contemporaine permet de mobiliser et de transférer rapidement l'information, d'accélérer le traitement des données. De plus, grâce aux simulations permises par certains logiciels, les élèves peuvent se rapprocher le plus possible d'une situation concrète, mesurant au mieux les conséquences de leurs décisions. Certains logiciels permettent de mieux visualiser les concepts théoriques sans pour autant recourir à l'expérience réelle qui est parfois coûteuse ou dangereuse. En effet, les technologies permettent de donner une perspective en trois dimensions (découverte des caractéristiques d'un cube en le faisant pivoter sous tous ses angles à l'écran).

[7] [DEPO87]

Cependant, il ne faut pas tomber dans les mêmes pièges tendus par l'arrivée des calculatrices. En effet, combien d'élèves réfléchissent-ils encore lorsqu'ils effectuent une division avant de sortir leur calculatrice? Savent-ils encore ce que signifie une table de multiplication? Aujourd'hui, le problème va se reposer mais d'une façon plus importante qu'autrefois. En effet, dans la mesure où l'ordinateur sera plus présent dans la société, il couvrira la plupart des matières enseignées. L'informatique est un outil mis à la disposition de l'élève, ne devant pas freiner ses initiatives. Ce dernier ne doit pas oublier le cheminement d'une résolution d'un problème pour arriver à sa solution. Il doit constamment être en état d'interpréter ce qu'il fait.

D. La possibilité de disposer d'un logiciel de qualité

Le manque de logiciels de qualité peut constituer un véritable frein à l'emploi de l'ordinateur dans l'enseignement.

L'ordinateur est-il capable de comprendre ce que l'élève ne comprend pas? Bien souvent, les logiciels d'apprentissage ne peuvent expliquer une matière que d'une ou de deux façons différentes. Si un élève ne comprend pas un concept théorique, le didacticiel lui renverra la même explication à chaque fois. D'ailleurs, très rares sont ceux qui envisagent différents types de solutions possibles pour résoudre un problème mathématique, par exemple. Dans le cadre d'un enseignement assisté par ordinateur, nous nous trouvons face à un espace de contraintes. De nombreux logiciels se basent la plupart du temps sur une réponse pré-conçue (c'est vrai ou c'est faux) et détectent difficilement des réponses partielles.

En ce qui concerne notre didacticiel, il s'inscrit dans la lignée des « mémoires-projets », c'est-à-dire qu'il est destiné à être poursuivi ultérieurement. Notre objectif est de proposer un maximum d'idées réalisables. La qualité de ce projet réside dans le fait qu'il propose différents outils pour aider l'élève dans la mise en équation du problème. Il laisse à l'apprenant une certaine marge de liberté dans la résolution du problème (cfr. 2.2.3.2 Le module de l'élève, page 34). Le professeur a, lui, la possibilité d'introduire les exercices de son choix, de créer un dictionnaire reprenant des termes propres aux mathématiques ainsi que des fiches d'aide pour les élèves, etc. (cfr. 2.2.3.1 Le module du professeur, page 29).

En conclusion

« La fée informatique ne changera pas l'élève en génie en herbe. Il y a des pédagogies à appliquer, des motivations à soutenir, des créativités à encourager, des exemples à donner, bref, l'activité pédagogique à poursuivre» [BIP56].

Il est certain que l'outil informatique ne remplacera jamais la pédagogie. Mais, elle peut enrichir la relation professeur-élève, privilégiant un enseignement individuel, adapté au niveau de connaissances de l'élève. L'ordinateur doit être, pour le professeur et ses élèves, une extension de leurs capacités de traitement. Mais, en aucun cas, ils ne doivent en devenir les esclaves. Nous nous sommes demandés s'il ne fallait pas innover en matière d'enseignement. Les technologies de l'information ne pourraient-elles pas apporter à l'enseignant, à l'élève, un nouveau moyen de communication, un nouveau point d'entente?

Beaucoup de projets sont en cours dans ce domaine. Mais, en attendant leur réelle concrétisation, le professeur doit continuer à transmettre à ses élèves le goût d'apprendre. Proche des élèves, tout en fixant des limites, l'enseignant doit constamment être à leur écoute, déterminer leurs besoins. Il est primordial qu'à la fin de leur formation, les élèves possèdent une capacité de flexibilité intellectuelle, considérant d'autres points de vue que les leurs. Ils doivent être capables de poursuivre leur route, sans l'enseignant, puisqu'ils auront appris à « apprendre ». Tel est le « challenge » le plus important pour le professeur.

La réalisation de tels projets devra tenir compte des réticences de certains professeurs, des difficultés des élèves à s'adapter à ce nouvel environnement de travail, des coûts que cela va engendrer, etc. Aussi, la création de classes 'pilotes' sera fort importante et permettra de nous éclairer sur les réactions, les attitudes des personnes face à ce renouveau en matière d'enseignement. L'essentiel est de bien tirer les conclusions de l'expérience en n'oubliant pas que, dans la plupart des cas, les personnes qui participent à cette épreuve sont des volontaires. Il faudra également tenir compte des réticences de certains professeurs à accepter la technologie dans leur enseignement, des difficultés des élèves à s'adapter à ce nouvel environnement de travail, etc.

Nous terminerons par un petit bout d'article trouvé dans le journal 'Le 7ème Soir', daté du 4 et 5 novembre 1995, p16.

« Les adultes de demain qui ne jongleront pas avec l'ordinateur seront les analphabètes du XXIe siècle. De ce point de vue, habituer dès aujourd'hui les plus jeunes à manipuler la souris et à passer d'un plan horizontal (une feuille ou un livre) à un autre vertical (l'écran) ne peut être que positif ».

Chapitre 2

L'analyse pédagogique

Introduction

Dans le chapitre précédent, nous avons fait plus amples connaissances avec nos utilisateurs potentiels. Il est certain « qu'enseigner » est une vocation. Il faut être à l'écoute des élèves, les comprendre pour mieux les aider à apprendre. Par ailleurs, le professeur a, lui aussi, ses exigences. Nous n'allons pas refaire le monde, juste améliorer la qualité de l'enseignement, d'une part, en offrant un outil pour la préparation des leçons du professeur; d'autre part, en proposant à l'élève « d'égayer » son apprentissage. Il est grand temps de nous plonger dans le coeur de notre sujet, à savoir, la réalisation d'un didacticiel de mise en équation et de résolution d'un problème mathématique.

Nous commencerons ce chapitre en présentant les objectifs pédagogiques que nous avons poursuivis. Ils se découpent en deux grands axes. Le premier sera de déterminer l'orientation de notre didacticiel; le second, de combiner les atouts d'un professeur et de notre didacticiel.

Ensuite, nous nous lancerons dans une analyse mathématique de notre projet. Nous déterminerons les prérequis nécessaires pour l'enseignement de cette matière. Nous nous permettrons de faire une remarque sur le schéma classique de résolution d'un problème du premier degré à une inconnue. Puis, à partir d'un exemple, nous réfléchirons sur l'élaboration de notre didacticiel. Nous verrons que ce dernier sera composé d'une phase d'élaboration des problèmes (le module du professeur) et d'une phase d'apprentissage (le module de l'élève).

Nous terminerons ce chapitre en proposant différents outils d'aide pour l'élève : le « diagnostiqueur » et l'aide sous forme de « mots clés ».

2.1 Les objectifs pédagogiques

2.1.1 Déterminer l'orientation de notre didacticiel

Avant de nous lancer dans l'élaboration du didacticiel, nous avons rencontré quelques professeurs de mathématiques de l'institut Ilon-S^{t} Jacques. Dans un ordre de difficulté croissante, nous avons essayé d'établir ensemble une table des différents types d'équations enseignés aux élèves de 2ème Rénové, de 3ème Technique et de 4ème Professionnelle. Le tableau ci-dessous reprend également le savoir-faire et les objectifs pédagogiques poursuivis par ces enseignants [ADGP82].

TYPE	SAVOIR-FAIRE	OBJECTIFS
Equation du premier degré à une inconnue $ax + b = 0$	♦ Reconnaître et résoudre une équation du premier degré à une inconnue en précisant les transformations effectuées. ♦ Reconnaître si un nombre est ou n'est pas solution d'une équation de ce type. ♦ Résoudre des problèmes se ramenant à la résolution d'une équation du premier degré à une inconnue.	♦ Utiliser les propriétés des opérations dans $\mathfrak{R}$. Ces propriétés se traduisent par des principes d'équivalence. ♦ Coder une phrase exprimée en français.
Système de deux équations du premier degré à deux inconnues $\begin{cases} ax + by + c = 0 \\ a' x + b' y + c' = 0 \end{cases}$	♦ Résoudre un système de deux équations linéaires à deux inconnues à coefficients numériques, admettant une solution unique. ♦ Vérifier si un couple de réels est solution d'un tel système. ♦ Résoudre des problèmes se ramenant à la résolution d'un système de deux équations linéaires à deux inconnues.	♦ Etendre les notions d'équations. ♦ Représenter graphiquement la solution d'un problème. ♦ Faire appel aux méthodes de résolution algébrique.

Comme nous vous l'avions présenté au cours de notre introduction, le sujet de ce mémoire est la réalisation d'un logiciel d'apprentissage de mise en équation et de résolution d'un problème mathématique. En se basant sur le tableau ci-dessus, nous avons décidé que le logiciel se limitera à l'apprentissage de mise en équation et de résolution de problème du premier degré à une inconnue.

Mais qu'entendons-nous par « problème mathématique »? Si nous reprenons la définition du Petit Robert, un problème serait une *« question à résoudre qui prête à discussion, dans une science »*. Pour les mathématiciens que nous avons pu rencontrer, la signification serait la suivante : « *Un problème mis en équation permet de poser la question : quels sont les nombres qui, remplaçant l'inconnue, transforment l'équation en une égalité? L'équation étant une égalité qui n'est vérifiée que pour certaines valeurs de l'inconnue* ». Ce didacticiel étant notamment destiné aux professeurs de mathématiques, à l'avenir, chaque fois que nous emploierons ce mot « problème », nous ferons référence à la définition donnée par ces mêmes professeurs.

2.1.2 Combiner les atouts d'un professeur et de notre didacticiel

Dans le cadre du cours d'algèbre, les élèves éprouvent souvent des difficultés importantes pour mettre un problème, présenté en français, sous la forme d'une équation. La résolution de cette équation revient à un « simple jeu d'écritures mathématiques », appliquant les règles de priorité, de distributivité, la réduction au même dénominateur ainsi que toutes les règles des égalités.

Aussi, le travail du professeur réside principalement dans le fait d'aider ses élèves à vaincre cette difficulté, et par la même occasion, de les réconcilier avec cette méthode de résolution d'un problème. Le professeur peut imaginer de nouvelles façons d'aborder une matière. De plus, le professeur sait repérer les points délicats d'une matière et il adapte ses explications aux erreurs commises par l'élève. Il est certain que tous les professeurs n'ont pas les mêmes qualités pédagogiques. Certains peuvent même manquer d'intuition, de créativité dans la présentation de leurs cours.

Le didacticiel permettra autant que possible au professeur de personnaliser ses exercices d'apprentissage. Par exemple, il lui sera laissé le choix de donner ou pas des explications sur telle donnée du problème, de lui soumettre ou pas une liste reprenant le choix d'inconnues, etc. Nous pouvons également concevoir que le didacticiel offre certaines possibilités d'illustrer la mise en équation du problème. Nous pouvons imaginer que, dans un problème de partage d'une somme, une des parts, par exemple l'inconnue, soit représentée par une bourse (icône). Il faudra veiller à ce que la tâche du professeur soit minimisée au niveau du temps de préparation de ses leçons.

Vis-à-vis de l'élève, nous espérons lui faire apprendre un certain nombre de notions liées à la mise en équation et à la résolution de problèmes mais aussi de lui permettre d'en découvrir par lui-même. Les notions incomprises peuvent être revues au rythme de l'élève et de façon plus détaillée. A cet effet, nous avons créé un outil d'aide dont le but est d'analyser les réponses d'un élève et de le guider en cas de besoin.

Principalement, notre travail consistera à reconstruire les mécanismes d'apprentissage employés par l'enseignant, en introduisant autant de méthodologie et de pédagogie qu'un professeur peut offrir à ses élèves.

2.2 L'analyse mathématique

2.2.1 Les prérequis

Avant d'aborder la question d'un problème du premier degré à une inconnue, il est nécessaire que les apprenants possèdent certains prérequis en calcul algébrique. Il s'agit de maîtriser les concepts suivants :

- Les opérations sur les nombres réels + - × ÷ ainsi que les exposants,
- Les règles des parenthèses,
- Les règles de priorité des opérations ( ) ^ × ÷ + -
- Le codage numérique ex.: le carré du triple de quatre ⇔ (3×4)^2
- Le codage littéral (1 ou 2 lettres) ex.: le double du produit de a et de b ⇔ 2×(a×b)

Il paraît évident que ces différentes notions citées ci-dessus devront être intégrées dans le didacticiel. Par exemple, le programme devra être capable d'appliquer les règles de priorité des opérations sur une expression littérale. Il devra donc distinguer les constantes simples (les nombres) et les coefficients de l'inconnue.

Nous accepterons que la valeur de l'exposant soit entière positive ou négative si elle s'applique aux nombres *Nbre* ($\forall$ n $\in$ Z, $\exists$ m $\in$ Nbre : m^n). En refusant que cette valeur soit un nombre rationnel, nous écartons la possibilité d'aborder des problèmes faisant appel aux racines d'un nombre. Il faut dire que ce type d'énoncé est peu fréquent si nous essayons de résoudre des problèmes du premier degré à une inconnue.

Pour cette même raison, nous n'aurons pas d'inconnue *Inc* élevée à une puissance supérieure à 1 ni inférieure à -1. Les seules valeurs admises seront -1, 0 et 1. Remarquons que si l'exposant vaut 0, alors le résultat de Inc^0 est le nombre 1; si l'exposant vaut 1, alors le résultat de Inc^1 est cette même inconnue. Diviser par une inconnue revient à multiplier par l'inconnue à la puissance -1.

Le codage est un prérequis fondamental pour la mise en équation d'un problème. En effet, l'apprenant doit être capable de « coder » en mathématique un énoncé écrit en français. Si nous reprenons l'énoncé ci-dessus, le carré du triple de quatre, l'apprenant doit être capable de coder la phrase sous forme mathématique, à savoir (3×4)^2. En aucun cas, il ne peut obtenir un résultat semblable à (3^2)×4.

Nous pourrions proposer, en guise d'introduction, une série d'exercices de codages numériques, introduits par le professeur qui estime que ce prérequis n'est pas suffisamment maîtrisé par ses élèves. Dans la mesure où l'apprenant devra se familiariser avec son environnement de travail (souris, sélection d'un mot de l'énoncé et mise en correspondance avec sa solution mathématique), nous pourrions prévoir quelques exercices introductifs au didacticiel. Bien entendu, cette introduction peut être négligée et le choix en est laissé au professeur.

Ces exercices pourraient être de différents types. L'élève devrait essayer de trouver,

- le codage d'une phrase donnée ex. : le carré du triple de quatre $\Leftrightarrow$?
- le décodage d'une expression mathématique ex.: $2\times(a\times b) \Leftrightarrow$?

Le second type d'exercices est plus difficile à corriger car nous pouvons obtenir une solution qualifiée de sémantiquement correcte mais qui pourrait être refusée par un programme strict. Nous entendons par « programme strict », un programme qui n'accepterait comme seule et unique réponse que la phrase suivante « le double du produit de a et de b », restreint à une correction syntaxique. Si nous laissons une part de liberté à l'élève pour rédiger la phrase, il peut taper n'importe quoi et ne jamais obtenir la bonne réponse même si elle est correcte au niveau sémantique : double du produit de a et b, produit de a et de b multiplié par deux. Pour éviter cela, il suffirait de proposer à l'élève une liste de mots comprenant des termes comme produit, somme, triple, carré, double, etc.

Nous voyons donc déjà, que nous allons être confrontés à la difficulté d'évaluer une expression introduite par l'apprenant. L'expression devra être correcte syntaxiquement et sémantiquement. Ceci fera principalement l'objet de notre préoccupation dans notre troisième chapitre. Préalablement, nous devons terminer notre analyse pédagogique.

2.2.2 Le schéma classique de résolution d'un problème du premier degré à une inconnue.

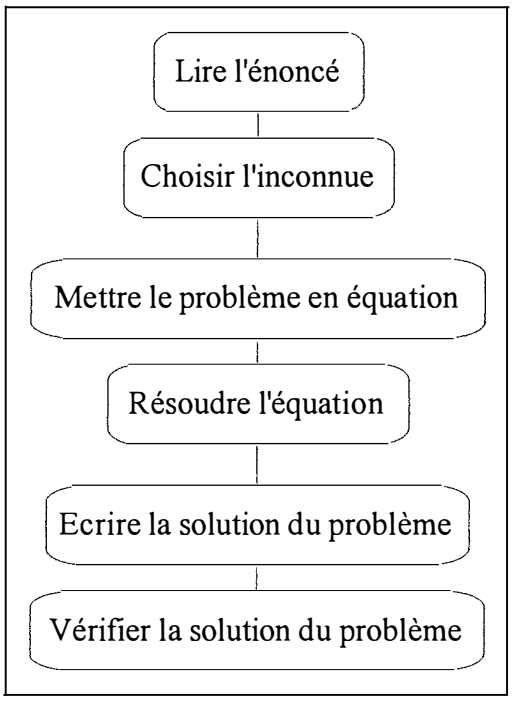


Figure 2.1 Schéma de résolution d'un problème du premier degré à une inconnue.

Le schéma de la page précédente reprend la méthode classique de résolution d'un problème du premier degré à une inconnue. Cette démarche est reprise dans de nombreux ouvrages [BOUT82], [ADMA82], etc.

Au niveau du didacticiel, nous ne souhaitons pas que l'élève soit « cloisonné » dans une procédure aussi canonique mais, au contraire, qu'il bénéficie d'une certaine marge de liberté. Aussi, au lieu de reproduire une technique de calcul, nous allons l'amener à raisonner sur chaque type de problème. Il n'est pas question d'appliquer une recette de cuisine! L'élève sera autorisé à commencer son exercice par l'étape de mise en équation. Nous pouvons concevoir qu'il revienne plus tard sur l'étape du choix de l'inconnue. A cet effet, toutes les étapes de résolution d'un problème du premier degré à une inconnue seront regroupées dans un même plan de travail, dans la fenêtre principale. Ainsi, l'élève pourra passer d'une étape à l'autre sans obligatoirement suivre un schéma prédéfini.

2.2.3 Une réflexion à partir d'un exemple

Le didacticiel se composera de deux grands modules : le module du professeur et le module de l'élève. Nous vous renvoyons au chapitre 4 pour une présentation plus explicite de l'interface du didacticiel. Elle respectera les règles ergonomiques énoncées dans le corpus de Jean Vanderdonckt[8].

2.2.3.1 Le module du professeur

L'objectif principal du module du professeur est de minimiser le temps de préparation de l'enseignant en lui faisant faire un minimum de manipulations. Nous allons partir de l'énoncé ci-dessous pour avoir quelques idées pour la réalisation du module du professeur. Les échanges entre le didacticiel (les messages apparaissant à l'écran) et le professeur, seront mis dans un encadrement. Les interventions du professeur seront indiquées en gras.

A. Enoncé du problème

Combien faut-il ajouter aux nombres 31 et 10 pour que le rapport des sommes obtenues soit égal à deux?

Le texte de l'énoncé doit apparaître à l'écran dans un style standard (sans mot en gras, en italique, etc.) pour ne pas induire l'élève en erreur.

B. Etape préliminaire : Traduction mathématique du langage courant

Cette étape préliminaire a pour objet de définir certains mots de l'énoncé. Elle n'est pas obligatoire si le professeur estime que l'énoncé n'a pas besoin d'explication.

[8] [VAND92]

Imaginons que le professeur souhaite préciser les mots « somme » et « rapport ». Tout d'abord, il sélectionne les mots dans l'énoncé à l'aide de la souris, c'est-à-dire qu'il les met en surbrillance. Ensuite, il valide ses sélections en appuyant sur le bouton de commande « Définir ». Une table apparaît à l'écran, composée de quatre colonnes « MOT », « SIGNIFICATION », « EXEMPLE » et « SYMBOLE MATH ».

MOT	SIGNIFICATION	EXEMPLE	SYMBOLE MATH
rapport	**Le rapport entre deux nombres est le quotient du premier nombre par le second**	**Le rapport entre 10 et 5 est égal à 2. Le rapport entre 5 et 10 est égal à ½**	/
somme	**Résultat d'une addition**	**La somme de 2 et de 5 est égale à 7**	+

Le didacticiel vérifie si les mots choisis par le professeur, se trouvent dans sa table TAB_EXPLICATION_MOTS permanente. Nous pouvons envisager qu'il s'agisse d'une petite base de données qui se construit au fil des énoncés. Cette base de données est modifiée chaque fois qu'on y apporte une modification (ajout de mot, modification de l'exemple, etc.). De plus, le professeur peut imprimer son contenu. Lors de la conception du didacticiel, nous prévoyons d'introduire une dizaine de mots (somme, différence, produit, quotient, etc.). L'identifiant de notre base de données est le champ « MOTS » sur lequel nous appliquons un tri par ordre alphabétique. L'avantage de cette base de données est qu'elle va permettre, d'une part, de favoriser la compréhension de l'élève lors de la lecture de l'énoncé; d'autre part, dans un souci de diminuer le temps de préparation du professeur, certains mots ne devront plus être définis puisqu'ils sont déjà repris dans notre base de données.

TAB_EXPLICATION_MOTS

MOT	SIGNIFICATION	EXEMPLE	SYMBOLE MATH
...	...	...	...
quotient	Résultat d'une division	Le quotient de 20 par 4 est égal à 5	/
...	...	...	...
rapport	*Le rapport entre deux nombres est le quotient[9] du premier nombre par le second*	*Le rapport entre 10 et 5 est égal à 2. Le rapport entre 5 et 10 est égal à ½*	/
...	...	...	...
somme	*Résultat d'une addition*	*La somme de 2 et de 5 est égale à 7*	+
...	...	...	...

[9] Le professeur fait référence à un autre mot qui pourrait être inconnu de l'élève. Heureusement, la base de données est dotée de quelques mots élémentaires.

Il se pourrait donc, qu'un des deux mots soit déjà repris dans la base de données. Les colonnes « SIGNIFICATION », « EXEMPLE » et « SYMBOLE MATH » associées à ce mot, sont remplies automatiquement par le didacticiel.

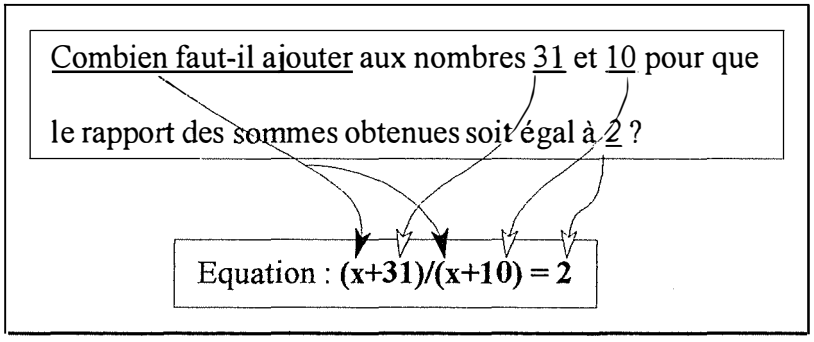
Cependant, le didacticiel permet au professeur de changer le contenu de ces colonnes et d'y ajouter sa propre explication et son exemple. Si le mot est inexistant, le professeur devra compléter les trois colonnes précitées (« SIGNIFICATION » et « EXEMPLE » et « SYMBOLE MATH »).

Il est utile de conserver les termes qui semblent importants aux yeux du professeur car ils sont sans doute la cause d'erreurs possibles dans la mise en équation. De plus, certains mots pourraient avoir une signification différente selon l'énoncé. Par exemple, « somme » dans notre énoncé n'a pas le même sens qu'une « somme de 1450 francs » dans un autre exercice. Nous les conserverons dans une nouvelle table liée à l'exercice TAB_MOT_ SYMBOLE.

TAB_MOT_SYMBOLE

MOT	SYMBOLE MATH
rapport	/
somme	+

C. Relation entre la mise en équation et l'énoncé du problème



Une fois que le professeur a introduit l'énoncé du problème et qu'il a ajouté éventuellement quelques commentaires, il met en équation le problème (expression mise en gras ci-dessous). Ensuite, il va établir des rapports entre chaque élément de l'équation ou bien un regroupement d'éléments (c'est-à-dire un ensemble significatif d'opérateurs et d'opérandes) et un ou plusieurs mots de l'énoncé. Nous avons représenté quelques-unes de ces correspondances par des flèches (noires s'il s'agit de l'inconnue, blanches s'il s'agit de nombres intervenant dans l'équation).

Concrètement, à l'écran, il existe une boîte de dialogue qui reprend la table des liaisons. Elle s'appelle *« Relation entre l'énoncé et la mise en équation du problème »*. Elle est constituée de trois colonnes « EQUATION », « ENONCE » et « COMMENTAIRES ». Le professeur positionne, par exemple, le curseur dans la colonne « EQUATION », il sélectionne, à l'aide de la souris, dans l'équation l(es) élément(s) qu'il souhaite analyser.

Ensuite, il positionne le curseur dans la colonne « ENONCE » et il sélectionne le ou les termes correspondants. Les deux premières colonnes sont remplies automatiquement par le didacticiel suivant les sélections effectuées par le professeur. La troisième colonne permet de préciser la relation entre l'énoncé et la mise en équation du problème.

RELATION ENTRE L'ENONCE ET LA MISE EN EQUATION DU PROBLEME

EQUATION	ENONCE	COMMENTAIRES
x	Combien faut-il ajouter	**L'inconnue du problème** **Le nombre à ajouter à 31 et à 10**
31 + x	ajouter 31	**Le premier terme du rapport**
10 + x	ajouter 10	**Le second terme du rapport**
(31+x)/(10 +x)	le rapport des sommes	**Le premier membre de l'équation**
= 2	égal à 2	**Le second membre de l'équation**
...	...	...

En fonction des renseignements fournis par le professeur, le didacticiel va pouvoir construire des tables qui serviront pour le module de l'élève.

TAB_LIAISONS_PROF

SEL EQUATION	SEL ENONCE	COMMENTAIRES	CONVERSION
?	?	*L'inconnue du problème* *Le nombre à ajouter à 31 et à 10*	---
?	?		31
?	?		10
?	?		2
...	...	...	...

Dans cette table, nous n'avons pu estimer les valeurs des deux premières colonnes. En effet, celles-ci contiennent la position où débute la sélection d'un mot ou d'une suite de mots (pour la colonne « SEL_ENONCE »), d'un ou de plusieurs éléments (pour la colonne « SEL_EQUATION ») ainsi que la longueur de la sélection. Il est possible que la sélection soit en fait la concaténation de plusieurs mots mis en surbrillance qui ne se suivent pas nécessairement. Il faudra tenir compte de cette remarque lors de l'analyse logicielle.

Une autre question se pose. Lorsque l'élève établira des relations entre l'énoncé et sa mise en équation du problème, que se passera-t-il si une de ses sélections est supérieure ou inférieure par rapport à la sélection du professeur? Il faudra déterminer la marge de liberté que le professeur souhaite laisser à l'élève. La solution que nous avons envisagée est de délimiter quelles sont les zones minimale et maximale de sélection autorisées.

Dans notre exemple, la sélection dans l'énoncé de l'inconnue du problème serait acceptée si l'élève sélectionne,

- au minimum la zone « Combien ajouter »,
- au maximum la zone « Combien faut-il ajouter aux nombres 31 et 10 ».

TAB_EQUATION_PROF

EQUATION	EQUATION CANONIQUE	SOL EQUATION	SOL PROBLEME
(x+31)/(x+10) = 2	-x +11 = 0	11	11

La seconde table que le didacticiel va créer s'appelle TAB_EQUATION_PROF. Elle reprend la mise en équation du problème, l'équation canonique, la solution de l'équation et celle du problème (ces deux dernières n'étant pas toujours identiques). Les deux dernières colonnes de la table sont en fait le résultat d'opérations internes, invisibles pour le professeur. L'équation canonique est le résultat obtenu suite à une procédure « Recherche_Equation_Canonique »[10]. A partir de cette équation canonique, le didacticiel résout l'équation. La solution trouvée est mise dans le champ « SOLUTION » de la table TAB_EQUATION_PROF.

D. Solution du problème

Bien souvent, les professeurs font en sorte que les problèmes qu'ils soumettent aux élèves proposent des opérations qui s'effectuent exactement. Cependant, il se pourrait que la réponse ne soit pas un nombre entier. Le didacticiel devra permettre une réponse sous forme de fraction, plus souvent employée qu'une réponse décimale.

SOLUTION DU PROBLEME

Le didacticiel vous propose la solution suivante :
Le nombre à ajouter à 31 et à 10 est 11.

Dans cet exemple, nous trouvons effectivement une réponse mais il se pourrait que nous ayons un problème avec un cas particulier. Ces deux cas sont :

- 0*X = 0, la solution de l'équation est indéterminée, le problème a une multitude de solutions,
- 0*X = 1, la solution de l'équation est impossible, le problème n'a pas de solution.

Quelques « clics » à droite et à gauche, et voilà le travail de notre cher professeur réduit à son minimum. Le reste (étapes de résolution de l'équation, écriture et vérification de la solution), c'est notre didacticiel qui s'en charge.

[10] Nous vous renvoyons au chapitre consacré à l'analyse logicielle.

2.2.3.2 Le module de l'élève

Entre la compréhension de l'énoncé et la résolution du problème, un dialogue va se créer entre l'élève et le système. Les échanges seront de deux types :

- des échanges d'informations reçues par le système en provenance de l'élève,
- des réponses du système à l'élève.

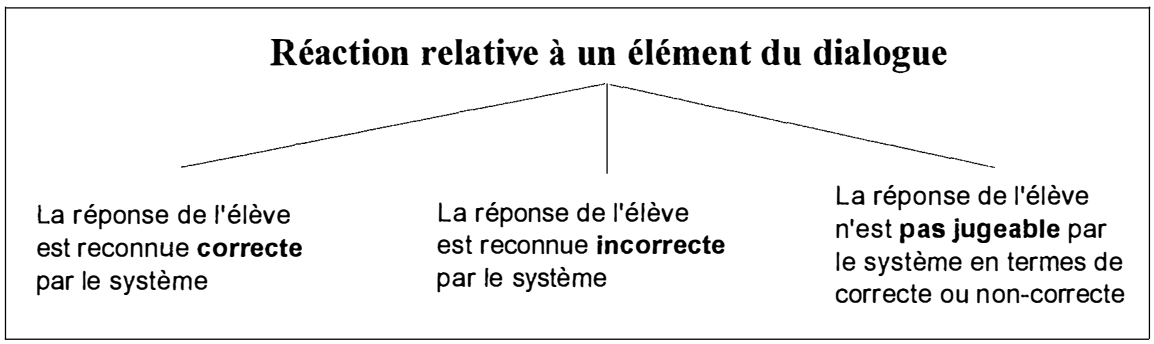
Le didacticiel pourrait être muni d'un chronomètre interne (timer). Après un certain temps, il va réagir en proposant de l'aide à l'élève, en vérifiant ce que l'élève a déjà fait, etc. Notre didacticiel ne possédera pas ce type de dispositif mais il offrira en revanche un outil d'aide tout aussi ingénieux (cfr. 2.3 Les outils d'aide, page 37).

Dans cette section 2.2.3.2, nous allons nous attarder aux différentes réactions possibles de notre système face à un élément de dialogue entre celui-ci et l'élève. Il existe différentes manières d'évaluer un élément de dialogue, une réponse fournie par l'élève.

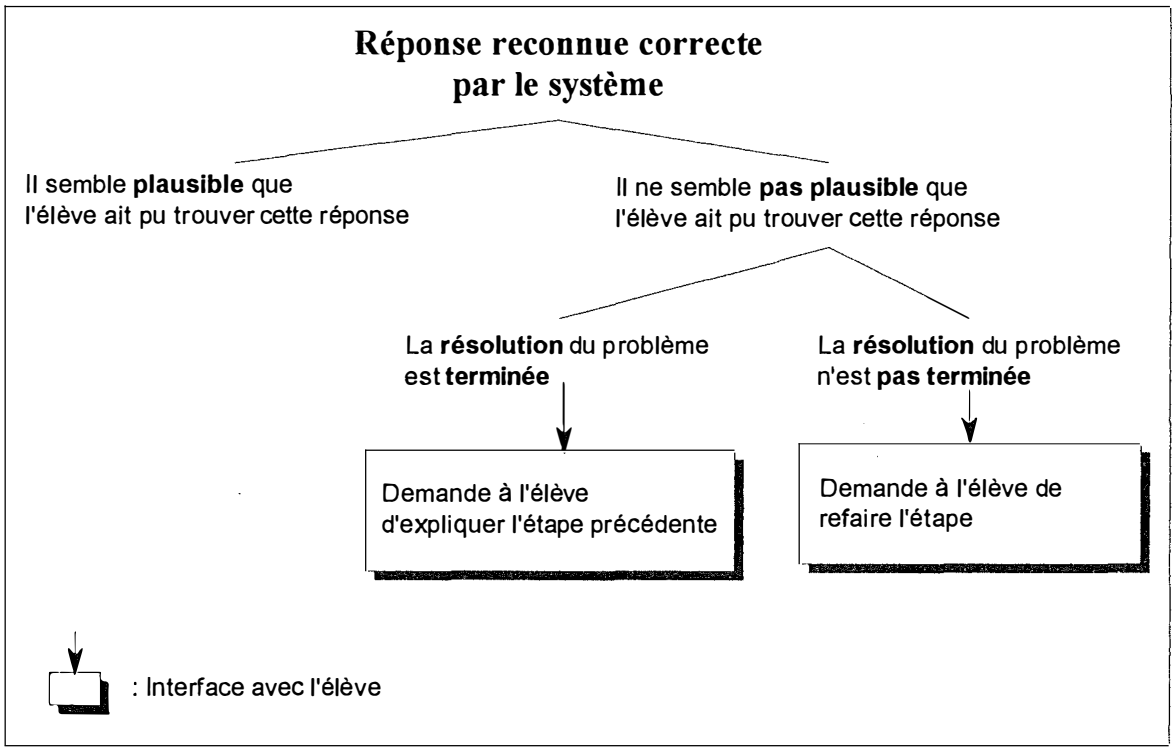
Premièrement, tous les éléments de dialogue pourraient être regroupés dans une base de données. Ils interviendraient en fonction des réponses prévues. Ce premier schéma d'analyse d'un élément de dialogue se retrouve dans les questions à choix multiples (QCM). Mais, cette solution serait fort fastidieuse pour le professeur car il devrait tout prévoir. N'oublions pas que notre but est de minimiser le temps de préparation de ses leçons.

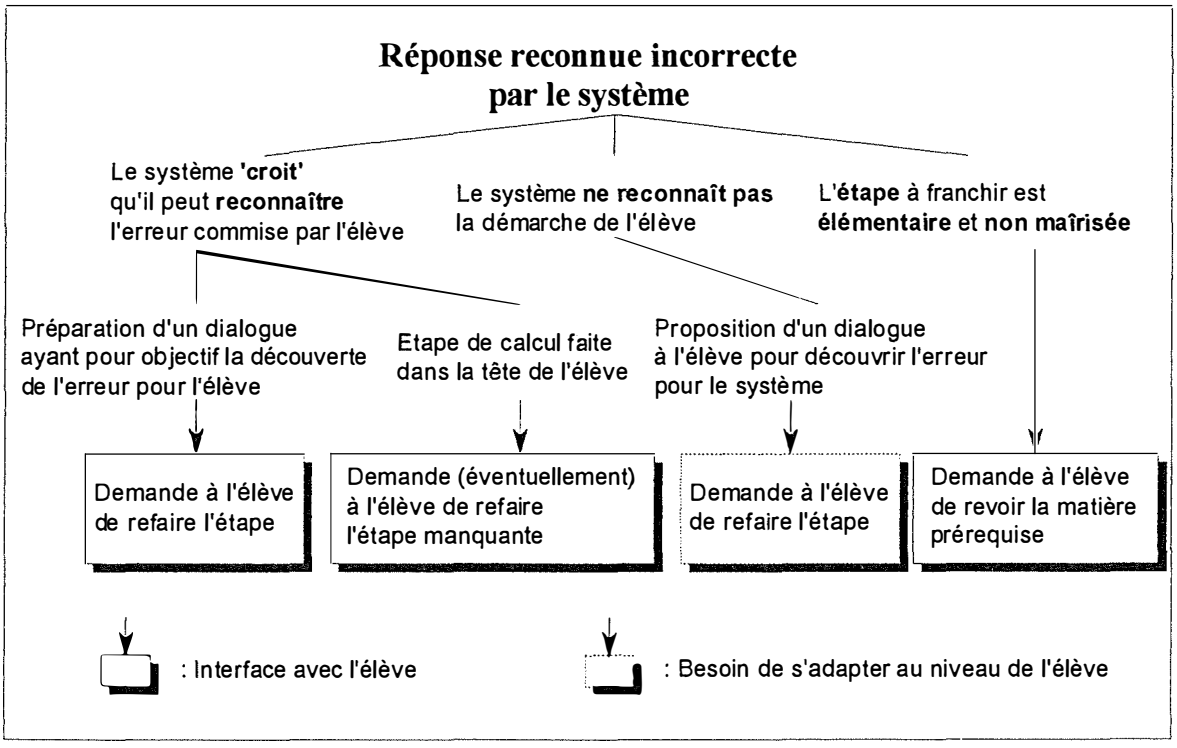
Deuxièmement, le logiciel aurait une certaine connaissance de la matière enseignée, c'est-à-dire qu'il se focaliserait sur une matière restreinte et déterminée. Ainsi, le didacticiel pourrait gérer lui-même toute une partie du dialogue. Cette seconde solution nous semble plus abordable et répond mieux à nos critères de spécification du didacticiel. Nous souhaitons que notre didacticiel s'adapte au niveau des connaissances de l'élève. Expert ou débutant dans le domaine de mise en équation et de résolution de problème, un élève doit pouvoir l'utiliser pour renforcer son apprentissage. Nous en reparlerons d'ailleurs au point 2.3 Les outils d'aide, page 37.

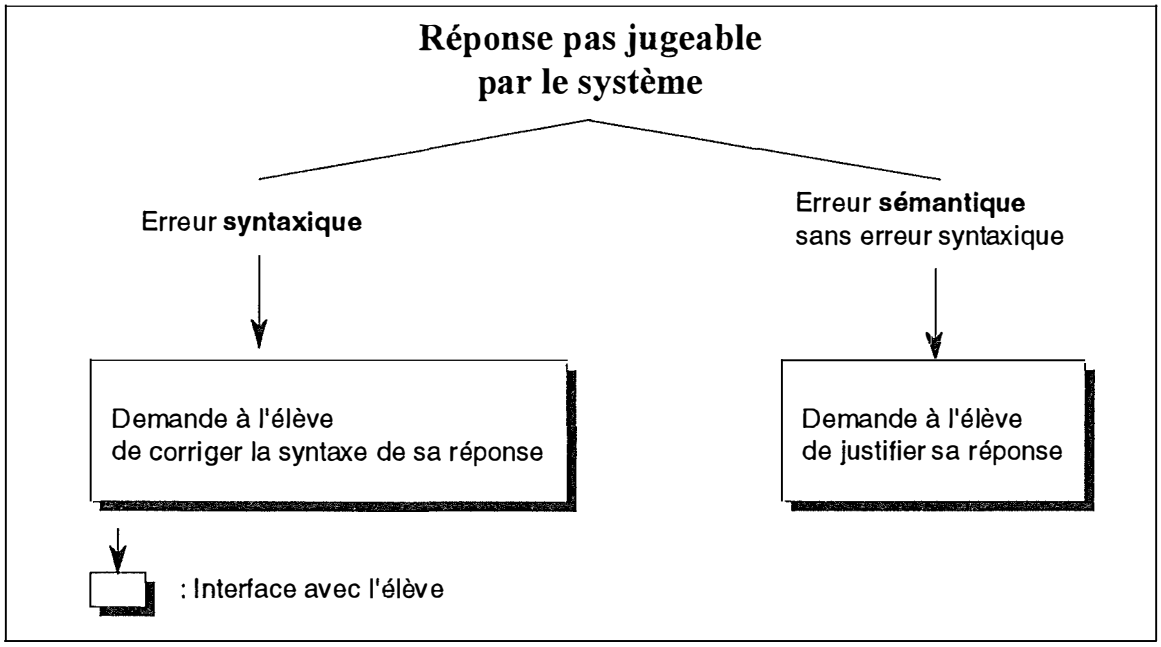
A présent, nous allons construire et commenter l'arborescence des différentes catégories de réponses que l'élève pourrait formuler.



Nous entendons par « *réponse correcte* », une réponse donnant un résultat juste avec une façon cohérente d'y arriver et qui est reconnue par le système.


Réponse reconnue correcte
par le système
Il semble **plausible** que
l'élève ait pu trouver cette réponse
Il ne semble **pas plausible** que
l'élève ait pu trouver cette réponse
La **résolution** du problème
est **terminée**
La **résolution** du problème
n'est **pas terminée**
Demande à l'élève
d'expliquer l'étape précédente
Demande à l'élève de
refaire l'étape
: Interface avec l'élève


Réponse reconnue incorrecte
par le système
Le système **'croit'**
qu'il peut **reconnaître**
l'erreur commise par l'élève
Le système **ne reconnaît pas**
la démarche de l'élève
L'**étape** à franchir est
élémentaire et **non maîrisée**
Préparation d'un dialogue
ayant pour objectif la découverte
de l'erreur pour l'élève
Etape de calcul faite
dans la tête de l'élève
Proposition d'un dialogue
à l'élève pour découvrir l'erreur
pour le système
Demande à l'élève
de refaire l'étape
Demande (éventuellement)
à l'élève de refaire
l'étape manquante
Demande à l'élève
de refaire l'étape
Demande à l'élève
de revoir la matière
prérequise
: Interface avec l'élève
: Besoin de s'adapter au niveau de l'élève



Nous pensons qu'il est important de bien comprendre cette arborescence reprenant les différentes catégories de réponses que l'élève pourrait formuler. En effet, nous sommes partis de celle-ci pour élaborer nos interventions auprès de l'élève telles que les boîtes de dialogue, les outils d'aide, etc. Et tout l'aspect didactique, les différents concepts et stratégies mis en jeu ont été réalisés par rapport à cette formalisation des réactions possibles de l'élève.

Certaines réactions du système seront malheureusement difficiles à concevoir. Comment pouvons-nous dire, ou plutôt comment le système peut-il savoir qu'il semble ***plausible*** que l'élève ait pu trouver cette réponse? Le système ***croit*** qu'il peut reconnaître l'erreur commise par l'erreur. L'élève a passé une étape de calcul, qu'est-ce qui ne nous dit pas qu'il l'a simplement oubliée? Le professeur connaît les compétences de chacun de ses élèves, il est capable de reconnaître un « bon » élève d'un élève plus « faible ». Mais, au fait, que veut dire « bon » ou « faible » élève? Le sens donné à ces qualificatifs peut varier d'un professeur à un autre. Tandis que notre didacticiel, lui, restera toujours objectif! Néanmoins, nous pourrions imaginer qu'à chaque élève, le professeur associe une note de connaissance. La valeur de cette note est égale à 1, s'il est plausible que l'élève ait pu trouver une réponse reconnue correcte par le système, égale à 0 dans le cas contraire.

Dans la mesure du possible, nous essayerons de faire réagir au mieux le système face aux différentes attitudes de l'élève.

2.3 Les outils d'aide

Bien souvent, les logiciels d'apprentissage ne peuvent expliquer une matière que d'une ou de deux façons différentes. Si un élève ne comprend pas un concept théorique, le didacticiel lui renverra la même explication à chaque fois. De plus, très rares sont ceux qui envisagent différents types de solutions possibles pour résoudre un problème mathématique, par exemple. Dans le cadre d'un enseignement assisté par ordinateur, nous nous trouvons face à un espace de contraintes. De nombreux logiciels se basent la plupart du temps sur une réponse préconçue (c'est vrai ou c'est faux) et détectent difficilement des réponses partielles.

2.3.1 Le « diagnostiqueur »

Le premier outil, que nous allons vous présenter, permettra à l'élève de se corriger, d'avancer dans la mise en équation, etc. Nous l'avons appelé le diagnostiqueur.

Il s'agit d'une boîte de dialogue qui reprend les différentes étapes de résolution d'un problème. Elle peut être soit en mode « Caché », soit en mode « Visible » en fonction du niveau de connaissance de l'élève. A tout moment, l'élève peut faire apparaître cette boîte de dialogue s'il se trouve en difficulté. Le didacticiel fait un petit bilan de la situation et associe à chaque étape une icône. Les différentes significations de l'icône sont :

- ☺ : L'étape est bien maîtrisée, continue!
- 💣 : C'est à ce niveau que se situe l'erreur, corrigeons-là!
- ⇧ : Tu ne peux pas avancer tant que l'étape précédente n'est pas résolue!

Nous pouvons également imaginer que chaque étape réussie s'illumine et que l'étape défectueuse clignote.

Dans l'exemple ci-dessous, nous voyons que les deux premières étapes, la lecture de l'énoncé et le choix de l'inconnue, ont été comprises. Cependant, la mise en équation du problème est incorrecte.

LE DIAGNOSTIQUEUR	
☺	1. Relis ton énoncé
☺	2. Choisis l'inconnue
💣	3. Mets le problème en équation
⇧	4. Résous l'équation
⇧	5. Donne la solution du problème

Les étapes de résolution d'un problème sont mises dans le menu offert par le « diagnostiqueur » (items numérotés de 1 à 5). Dès qu'une étape est choisie, diverses fonctions sont offertes à l'élève :

MA. Le didacticiel propose à l'élève de le guider dans cette étape, pas à pas;
MB. Le didacticiel propose à l'élève de poursuivre la résolution du problème sans son aide. L'élève, en voyant le diagnostic du didacticiel, souhaite poursuivre seul car il sait où il a commis une erreur;
MC. Le didacticiel renvoie vers une aide globale organisée sous forme de mots-clés (liens hypertextes).

2.3.1.1 Quelques exemples d'applications du « diagnostiqueur »

Reprenons notre arborescence des différentes catégories de réponses que l'élève pourrait formuler. A l'aide de quelques exemples, nous allons vous expliquer le fonctionnement du « diagnostiqueur ». L'outil d'aide est mis en mode « Visible ».

La réponse est reconnue incorrecte - l'étape à franchir est élémentaire et non maîtrisée

Dans la mesure où l'élève a du mal à commencer la mise en équation du problème, il faut le guider pas à pas (méthode MA) en commençant par relire l'énoncé (étape 1). Si l'élève se sent capable de poursuivre seul, par exemple, après la mise en équation, il pourra choisir une méthode MB pour résoudre l'équation et la suite des opérations. A tout instant, il peut consulter l'aide globale.

La réponse est reconnue incorrecte - le système croit qu'il peut reconnaître l'erreur commise par l'élève - préparation d'un dialogue permettant de découvrir l'erreur

Le didacticiel établit un diagnostic en comparant ce que l'élève a écrit avec ses différentes tables créées précédemment avec le module du professeur pour cet énoncé. Ensuite, lorsqu'il aura détecté l'étape défectueuse, il propose les trois méthodes possibles (MA, MB, MC) à l'élève qui lui permettront de continuer. Par exemple, l'élève a oublié une donnée du problème (renvoi à l'étape 3).

Exemple basé sur l'énoncé du problème de la page 29 :
$x/(x+10) = 2$ au lieu de $(x+31)/(10+x) = 2$

La réponse est reconnue incorrecte - le système ne reconnaît pas la démarche de l'élève

Le didacticiel envoie un message d'incompréhension, d'interrogation à l'écran car il n'a pas trouvé de correspondance avec un quelconque élément de ses tables. Il propose à l'élève de choisir une étape qu'il maîtrise. Si, à la fin de cette étape, le didacticiel reconnaît une donnée qui serait en rapport avec ses tables, il demandera à l'élève de passer à l'étape suivante. Par contre, si l'élève propose toujours une réponse non-mathématique ou si l'étape n'est pas correcte, le didacticiel renverra l'élève à l'étape précédente et ainsi de suite, jusqu'à ce qu'il arrive à l'étape 2. A partir de ce moment-là, le didacticiel guidera l'élève pas à pas (méthode MA) en commençant par la relecture de l'énoncé (étape 1).

La réponse est reconnue correcte - il ne semble pas plausible que l'élève ait pu la trouver - la résolution du problème est terminée

Dans ce cas-ci, le didacticiel renverra l'élève à l'étape 3 (mise en équation). Si celle-ci s'avère juste, le didacticiel demandera à l'élève d'effectuer les étapes suivantes. Sinon, il faudra que l'élève recommence à partir de l'étape 1.

La réponse est reconnue correcte - il ne semble pas plausible que l'élève ait pu la trouver - la résolution du problème n'est pas terminée

Il existe divers types de réponses partielles. Les exemples qui suivent sont basés sur l'énoncé du problème de la page 29.

- L'élève n'a pas neutralisé le coefficient de X (renvoi à l'étape 4)
  ex. : -x = -11
- L'élève n'a pas simplifié une expression, une fraction (renvoi à l'étape 4)
  ex. : x = (-11/-1)
- L'élève n'a pas distingué la solution de l'équation de la solution du problème. L'élève doit pouvoir interpréter les deux cas particuliers cités précédemment (cfr. page 33).

La réponse n'est pas jugeable par le système - Erreur syntaxique

Dans le chapitre 3, consacré aux concepts et aux stratégies mis en jeu, nous verrons que le didacticiel est doté d'un analyseur syntaxique. Par conséquent, il pourra vérifier si la syntaxe d'une expression mathématique, dans notre cas, d'une équation, est correcte. Par exemple, l'élève a oublié un membre de l'équation (renvoi à l'étape 3).

Exemple basé sur l'énoncé du problème de la page 29 :
(31+x)/(x+10) au lieu de (31+x)/(x+10) = 2

La réponse n'est pas jugeable par le système - Erreur sémantique sans erreur syntaxique

Nous présenterons également, dans le chapitre 3, un analyseur sémantique. Ce dernier nous permettra de vérifier si la sémantique d'une expression mathématique, dans notre cas, d'une équation, est correcte. Par exemple, l'élève a mal interprété l'énoncé du problème (renvoi à l'étape 1).

Exemple basé sur l'énoncé du problème de la page 29 :
(10+x)/(31+x) = 2 au lieu de (31+x)/(10+x) = 2

2.3.1.2 Le détail des trois premiers items du « diagnostiqueur » - Méthode MA

Pour rappel, le diagnostiqueur reprend les différentes étapes de résolution d'un problème. A tout moment, l'élève peut faire apparaître cette boîte de dialogue s'il se trouve en difficulté. Le didacticiel fait un petit bilan de la situation et associe à chaque étape une icône. Nous allons détailler les trois premières étapes selon la méthode MA, c'est-à-dire que le didacticiel propose à l'élève de le guider pas à pas. Pour les deux dernières étapes, la résolution de l'équation et la solution du problème, l'élève en difficulté peut se référer à l'aide sous forme de « mots-clés ».

A. Relis ton énoncé

Cette première étape a deux objectifs. Premièrement, elle incite l'élève à relire l'énoncé attentivement et à repérer les mots qui lui posent une difficulté au niveau de la traduction mathématique du langage courant. Si nous reprenons l'exemple de la page 29, les deux mots qui pourraient poser un problème de compréhension sont « rapport » et « somme ». Deuxièmement, le didacticiel peut donner des explications sur certains mots de l'énoncé.

B. Choisis l'inconnue

Comment choisir l'inconnue du problème, que représente-t-elle? Dans cette étape, nous proposons quelques types de questions permettant à l'élève de découvrir ce que représente l'inconnue dans l'équation. Les échanges entre le didacticiel (les messages apparaissant à l'écran) et l'élève, seront mis dans un encadrement. Les interventions de l'élève seront indiquées en gras.

COMMENT PEUT-ON TROUVER L'INCONNUE?

- Que demande-t-on de trouver?
- Que doit-on rechercher?
- Sur quoi pose la question du problème?

Si ces questions ne suffisent pas pour trouver l'inconnue du problème, nous lui proposerons, à l'élève, une liste d'éléments rédigée par le professeur (afin de diminuer l'intervention de l'élève par le biais du clavier). Dans cette liste, nous retrouvons des données qui peuvent ou ne peuvent pas représenter l'inconnue. L'élève sélectionne la représentation dans la liste de sélection.

Le didacticiel va aller vérifier dans ses tables si ce que l'élève a choisi dans la liste ou a sélectionné à l'aide de la souris dans l'énoncé, correspond avec ce que le professeur[11] a, lui aussi, introduit préalablement. Dans le cas de notre exemple, il faut que

- Rechercher l'enregistrement i où TAB_LIAISONS_PROF[i].CONVERSION = ' '[12];
  Si [TAB_LIAISONS_PROF[i].SEL_ENONCE = ce qui a été sélectionné par l'élève à l'aide de la souris dans l'énoncé] **et** [TAB_LIAISONS_PROF[i].SEL_EQUATION = ce qui a été sélectionné par l'élève à l'aide de la souris dans son équation]
  Alors Mettre à jour la table TAB_LIAISONS_ELEVE
  Afficher à l'écran TAB_LIAISONS_PROF[i].COMMENTAIRES;
  OK:=true
  Sinon Donner une deuxième chance à l'élève de se corriger[13]

[11] Cfr. 2.2.3.1 Le module du professeur, page 29.

[12] Le champ 'CONVERSION' de la table TAB_LIAISONS_PROF est la transformation d'une chaîne de caractères (c.-à-d. la représentation d'un nombre dans l'équation) en une valeur numérique. Etant donné que l'inconnue n'est pas un nombre, le champ est initialisé à un caractère blanc.

[13] Si l'élève, après un second essai, ne trouve pas la solution, le logiciel la lui donnera avec d'éventuelles explications.

C. Mets le problème en équation

Maintenant, nous devons traduire par des symboles mathématiques les relations entre les données et les inconnues fournies par l'énoncé du problème. La première étape consiste à rechercher dans l'énoncé du problème une phrase qui exprime l'égalité de deux expressions, c'est-à-dire que l'élève doit trouver les deux membres de l'équation. Ensuite, l'élève devra exprimer les deux membres qui entrent dans la relation d'égalité au moyen des nombres donnés et de l'inconnue X, comme si cette dernière était déjà connue. On obtient ainsi l'équation du problème. Selon le problème, l'approche envisagée sera différente.

Partons de l'énoncé de la page 29.

METTONS LE PROBLEME EN EQUATION

Quelles sont les différentes données que nous trouvons dans l'énoncé dont nous ne nous sommes pas encore servis? **31, 10, 2**

L'élève sélectionne les différentes données dans l'énoncé, à l'aide de la souris. Le didacticiel va vérifier dans le champ « SEL_ENONCE » de la table TAB_LIAISONS_PROF si elles existent.

RELATION ENTRE L'ENONCE ET LA MISE EN EQUATION DU PROBLEME

EQUATION	ENONCE	COMMENTAIRES
x	Combien faut-il ajouter	**L'inconnue du problème** **Le nombre à ajouter à 31 et à 10**
31 + x	ajouter 31	**Le premier terme du rapport**
10 + x	ajouter 10	**Le second terme du rapport**
(31+x)/(10 +x)	le rapport des sommes	**Le premier membre de l'équation**
= 2	égal à 2	**Le second membre de l'équation**
...	...	...

EQUATION : **(31+x)/(10+x) = 2**

Si ces données sont correctes, elles sont affichées, à l'écran, dans la table *« Relation entre l'énoncé et la mise en équation du problème »*. Le champ « SEL_ENONCE » de la table TAB_LIAISONS_ELEVE est mis à jour. Ensuite, l'élève sélectionne les différents éléments de l'équation à l'aide de la souris et établit des liens avec les mots de l'énoncé. Le didacticiel va vérifier dans sa table TAB_LIAISONS_PROF si ces liens correspondent à ceux que le professeur avait sélectionnés.

La mise en équation est vérifiée, entre autres, en comparant le contenu du champ « EQUATION_CANONIQUE » de la table TAB_EQUATION_PROF avec celui de la table TAB_EQUATION_ELEVE. Si elles ne sont pas équivalentes, l'élève peut ne pas avoir compris un mot de l'énoncé (renvoi à la table TAB_MOT_SYMBOLE) ou une relation (renvoi aux explications du professeur - TAB_LIAISONS_PROF.COMMENTAIRES).

2.3.2 L'aide sous forme de « mots-clés »

L'aide sous forme de « mots-clés » ressemblera fortement à l'aide proposée par le logiciel Windows (« Topics »). En effet, cette aide contiendra des liens « hypertextes ». Dans le module du professeur, nous envisageons de créer un éditeur de liens « hypertextes ». Ainsi, le professeur pourra réaliser ses propres pages d'aide.

Nous vous proposons un exemple d'aide, basé sur ce principe, lié à la résolution d'équations à une inconnue dans R.

Résolution d'équations à une inconnue dans R

■ Exemple

Règles de simplifiabilité des opérations dans R appliquées aux équations
Définitions
Cas extrêmes

Règles de simplifiabilité des opérations dans R appliquées aux équations

■ Exemples

Règle 1
Lorsqu'on ajoute un même réel aux membres d'une équation, on obtient une équation équivalente à la première.

a, b, c étant des nombres quelconques,
si $a = b$ *alors* $a + c = b + c$ *quel que soit c et réciproquement*

Règle 2
Lorsqu'on multiplie par un même réel non nul les deux membres d'une équation, on obtient
une équation équivalente à la première.

a, b, c étant des nombres quelconques,
si $a = b$ *alors* $a \times c = b \times c$ *quel que soit c et réciproquement*

Définitions

Définition 1
Une *équation dans R à une inconnue* est une phrase codée sous forme d'égalité qui pose la question : « Quels sont les réels qui, mis à la place de l'inconnue, donnent une égalité? ».

Définition 2
Une *solution d'une équation dans R à une inconnue* est un réel qui transforme l'équation en une égalité, c'est-à-dire un réel qui vérifie l'équation.

Cas extrêmes

Cas 1
Dans $\mathfrak{R}$, l'équation 0*X=0 admet tout réel comme solution : S = $\mathfrak{R}$.
L'équation est dite « indéterminée ».

Cas 2
Dans $\mathfrak{R}$, l'équation 0*X=1 n'admet aucun réel comme solution : S =$\varnothing$ ou S = {}.
L'équation est dite « impossible ».

Exemples d'applications des règles de simplifiabilité

Exemple 1
Lorsqu'un membre d'une équation est une somme, tu élimines un terme de ce membre en ajoutant son opposé aux deux membres.

ex.: x+3=5
*x+3**-3**=5**-3***
x=2

Exemple 2
Lorsqu'un membre d'une équation est un produit dont un des facteurs est non nul, tu élimines ce facteur en multipliant les deux membres par son inverse.

ex.: -3x=5
$(-\frac{1}{3}) \times (-3x) = (-\frac{1}{3}) \times 5$
$x = -\frac{5}{3}$

Exemple de résolution d'une équation à une inconnue

$$5\times(x-2)=3-(x-\frac{1}{2})$$

1) Effectuer en éliminant les parenthèses	$5x-10=3-x+\frac{1}{2}$	Calculs dans $\Re$
2) Mettre au même dénominateur	$\frac{10x-20}{2}=\frac{6-2x+1}{2}$	Calculs dans $\Re$
3) Supprimer le dénominateur commun	$10x - 20 = 6 -2x +1$	**Règle 2 de simplifiabilité**
4) Rassembler les termes en x dans un membre et les termes indépendants dans l'autre.	$10x+2x=6+20+1$	**Règle 1 de simplifiabilité**
5) Ramener à la forme ax=b	$12x = 27$	Calculs dans $\Re$
6) Neutraliser le coefficient de x	$x = \frac{27}{12}$	**Règle 2 de simplifiabilité**
7) Simplifier la réponse	$x=\frac{3\times 9}{3\times 4}$	Calcul fractionnaire
8) Exprimer l'ensemble S des solutions	$S = \{\frac{9}{4}\}$	

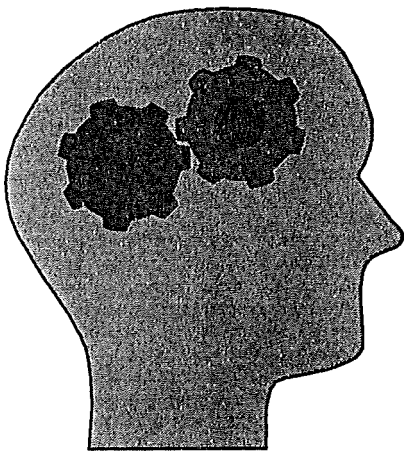
Conclusion

Il n'existe pas une, mais bien mille et une façons d'enseigner. Une méthode peut s'avérer tantôt efficace, tantôt catastrophique. Sa réussite dépendra de la relation établie entre le professeur, l'élève et la matière enseignée. Nous n'avons pas voulu que ce didacticiel soit juste un outil mécanique, fermant la porte aux interventions de l'enseignant. Il nous a semblé important que le professeur, ayant une réelle antériorité par rapport à la matière, puisse intervenir d'une façon dynamique. C'est dans ce but que nous avons voulu développer le module du professeur. Notre module de l'élève, lui, consiste à aider l'élève à approcher la matière, afin qu'il puisse la comprendre, la connaître à son rythme en fixant ses objectifs d'apprentissage.

Dans ce chapitre, nous avons essayé de donner une esquisse du travail que nous voudrions réaliser. Nous avons mis l'accent sur trois grandes idées : la réalisation de deux modules (celui de l'élève et celui du professeur) et d'outils d'aide. Le cadre de notre projet étant délimité, nous allons pouvoir passer aux concepts et aux stratégies mis en jeu dans ce mémoire.

Chapitre 3

Les concepts et les stratégies mis en jeu



Introduction

L'optique de notre didacticiel n'est pas d'exposer simplement une nouvelle matière en l'affichant à l'écran. Nous incitons les élèves à apprendre une leçon en leur proposant une série d'exercices pour lesquels ils doivent trouver une solution. Cependant, il faut que le didacticiel soit capable d'interpréter chacune de leur réponse et d'en vérifier leur exactitude. Nous avions déjà présenté quelques stratégies dans le chapitre précédent, telles qu'offrir une liste de réponses potentielles à l'élève pour choisir l'inconnue du problème.

Mettre un problème en équation et résoudre cette équation exigent que le didacticiel utilise une tout autre stratégie. La représentation et la manipulation d'expressions mathématiques ne peuvent pas se restreindre à de simples « clics » dans une liste déroulante. Nous devons prévoir des mécanismes qui permettent d'analyser, de comparer la solution de l'élève avec celle du professeur.

Lorsque l'élève mettra le problème sous forme d'équation, nous devrons vérifier si l'expression qu'il a introduite au clavier respecte une certaine structure syntaxique, c'est-à-dire qu'elle soit conforme à notre système de représentation d'une équation. Il faudra également vérifier la sémantique via les correspondances que l'élève a réalisées entre l'énoncé et l'expression.

Ensuite, l'élève essayera de résoudre l'équation qu'il a précédemment introduite, c'est-à-dire l'expression que nous venons d'analyser syntaxiquement et sémantiquement. Pour résoudre cette équation, l'élève entrera de nouvelles expressions mathématiques qui devront lui permettre de trouver la solution de l'équation. Nous vérifierons la syntaxe de chacune de ces expressions. Puis, nous contrôlerons leur sémantique en comparant leur arbre de décomposition syntaxique avec celui de l'expression précédemment analysée.

Nous distinguerons trois phases d'analyse, la première (analyse lexicale) consiste à découper l'expression introduite par l'utilisateur en petites entités (symboles terminaux). La deuxième phase (analyse syntaxique) consiste à expliciter la structure de l'expression sous forme d'un arbre, appelé arbre de syntaxe, chaque noeud de cet arbre correspond à un opérateur et ses fils aux opérandes sur lesquels il agit. La troisième phase (analyse sémantique) entreprend la description du sens de l'expression. Les deux premières phases seront effectuées par le module de l'analyseur syntaxique.

3.1 L'analyseur syntaxique

3.1.1 La composition d'une grammaire

« Tout langage de programmation a des règles qui prescrivent la structure syntaxique des programmes bien formés » [LERO94]. L'expression mathématique, introduite par l'utilisateur, doit, elle aussi, respecter une certaine structure syntaxique.

Notre grammaire est constituée d'une *collection de catégories syntaxiques*. Nous définissons une *catégorie syntaxique* comme étant la concaténation de symboles terminaux avec des éléments d'une ou d'autres catégories syntaxiques. Les *règles syntaxiques* sont des transformations sur l'ensemble des catégories syntaxiques. Elles permettent d'établir des relations entre celles-ci et définissent ainsi la syntaxe de notre grammaire.

3.1.1.1 Une grammaire définie selon Chomsky

Les règles sont de la forme,

$S \rightarrow \rho$ avec $S \in V_N$

ρ : chaîne non vide sur le vocabulaire V

et $V = V_T \cup V_N \cup \Lambda$

V_T : vocabulaire terminal

V_N : vocabulaire non terminal[14]

Λ : chaîne vide

« Elles permettent de réécrire une chaîne $\varphi = \varphi_1 S \varphi_2$ sous la forme $\psi = \varphi_1 \rho \varphi_2$ ce que Chomsky note $\varphi \rightarrow \psi$ en utilisant le même symbole $\rightarrow$ qui signifie ici 'se réécrit' » [BRAS65].

Si nous définissons une grammaire d'après Chomsky, *« il s'agira d'une grammaire de génération qui permet de construire les chaînes du langage correspondant à partir d'un élément distingué, S, du vocabulaire non terminal, élément que nous appellerons axiome » [BRAS65].*

La suite de dérivations d'une chaîne ω est, par définition, *« une séquence de chaînes : $\varphi_1, \varphi_2, ..., \varphi_n$ $n \geq 1$ telle que $\varphi_1 = S$, $\varphi_n = \omega$ et $\varphi_i \rightarrow \varphi_{i+1}$ $(1 \leq i < n)$, c'est-à-dire que φ_{i+1} est une réécriture de φ_i » [BRAS65].*

EXEMPLE

Soient le vocabulaire terminal $V_T = \{x, y\}$

le vocabulaire non terminal $V_N = \{S, X, Y\}$

les règles syntaxiques suivantes constituant la grammaire

1. $S \rightarrow xS$ 2. $S \rightarrow XYy$ 3. $X \rightarrow Xx$ 4. $X \rightarrow x$ 5. $Y \rightarrow y$

[14] Les éléments de ce vocabulaire tiennent lieu de variables intermédiaires servant à engendrer des catégories syntaxiques.

Une suite de dérivations de la chaîne ω = xxyy sera
φ_1 = S
φ_2 = xS en appliquant la règle 1
φ_3 = xXYy en appliquant la règle 2
φ_4 = xxYy en appliquant la règle 4
φ_5 = xxyy en appliquant la règle 5

Lors de l'analyse syntaxique, la question que nous allons nous poser est de savoir si une expression mathématique introduite par l'utilisateur est une suite de dérivations en référence à notre grammaire que nous aurons définie.

Si, à toute expression, nous associons un formalisme mettant en évidence une suite de dérivations qui conduit depuis l'axiome S à cette expression, nous pourrons dire que nous avons donné une structure à cette expression.

3.1.1.2 La forme normale de Backus [REEV67]

Nous allons utiliser plutôt la forme normale de Backus (B.N.F.) pour définir nos différentes règles syntaxiques. *« La notation B.N.F est une écriture condensée des règles » [BRAS65].* Si nous reprenons les règles syntaxiques énoncées ci-dessus (cfr. 3.1.1.1 Une grammaire définie selon Chomsky, page 48), en notation de Backus, nous aurons,

< S > ::= x < S > | < X > < Y > y
< X > ::= < X> x | x
< Y > ::= y

La notation B.N.F. représente la grammaire d'un langage comme un ensemble de définitions de structures grammaticales. Chaque définition, c'est-à-dire chaque règle syntaxique, se compose de trois éléments,

- le nom de la structure que nous définissons, c'est-à-dire la catégorie syntaxique (partie gauche), mis entre *braquets*;
- le symbole ::= qui signifie « est défini par ». Il va remplacer le symbole $\rightarrow$ dans les règles $\Sigma \rightarrow \rho$. Il s'agit en fait d'un méta-symbole car il appartient à un méta-langage et non au langage dont on a défini la grammaire;
- l'expression (liste d'alternatives) qui spécifie la forme autorisée de la structure (partie droite).

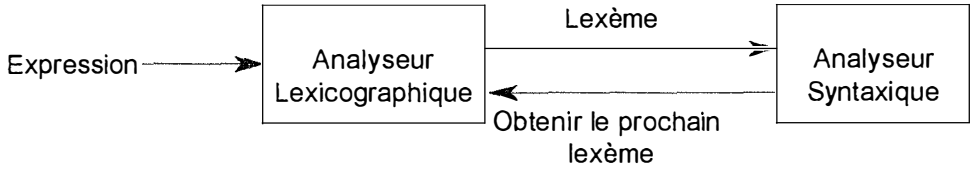
Lorsque des règles ont la même partie gauche, nous ne répétons pas celle-ci, mais nous écrivons les diverses parties droites successivement séparées par le méta-symbole |. Chaque alternative est une liste d'éléments écrits l'un après l'autre. Le sens d'une alternative est obtenu en lisant « est suivi de » entre chaque paire adjacente d'éléments.

Un élément est

- une constante, c'est-à-dire tout symbole appartenant à l'alphabet du langage défini (vocabulaire terminal);
- ou bien, une variable qui a la forme du nom d'une structure et représente toute forme permise définie pour cette structure.

3.1.2 Le rôle de l'analyseur lexicographique

Le rôle de l'analyseur lexicographique est de décomposer l'expression à analyser en unités lexicales appelées *lexèmes*. Chacun d'eux est transmis à l'analyseur syntaxique.



Dans un premier temps, nous effectuerons donc une analyse lexicographique qui va nous permettre de faire le premier lien entre la représentation de l'équation (ligne récemment introduite au clavier par l'utilisateur) et son *image* dans notre système de représentation. En effet, le rôle de notre analyseur lexicographique est de transformer l'expression mathématique en une autre, appartenant à la grammaire que nous définirons. L'expression entrée par l'utilisateur va être découpée séquentiellement afin de faire apparaître des symboles terminaux. Dès lors, nous pourrons détecter les caractères incompatibles avec notre système de représentation.

A la fin de cette étape, nous obtenons une suite de symboles terminaux. Cette suite correspond à l'image de l'expression mathématique introduite par l'utilisateur dans notre système de représentation. Il faudra tester si l'enchaînement de ces symboles terminaux est syntaxiquement correct.

EXEMPLES

Soient le vocabulaire terminal V_T = { x, =, y, (,) }
le vocabulaire non terminal V_N = { E, M, V }
les règles syntaxiques suivantes constituant la grammaire

< E > :: = < M > = < M >
< M > :: = (< E >) | y | x

Exemple n°1

Ligne introduite par l'utilisateur : x = y
Analyseur lexicographique : Tous les caractères sont des symboles terminaux reconnus par notre système de représentation.

Image de la ligne dans notre système de représentation : { x , =, y }

Exemple n°2

Ligne introduite par l'utilisateur : x = ?
Analyseur lexicographique : Le caractère « ? » n'est pas compatible avec notre système de représentation.

Pas d'image

Exemple n°3

Ligne introduite par l'utilisateur: x = y)
Analyseur lexicographique : Tous les caractères sont des symboles terminaux reconnus par notre système de représentation.
Image de la ligne dans notre système de représentation : { x, =, y,) }

L'exemple n°2 montre l'utilité d'analyser lexicographiquement l'expression mathématique au départ. En effet, il est inutile de poursuivre notre analyse que ce soit au niveau sémantique ou au niveau syntaxique. Par contre, l'exemple n°3 montre que l'analyse lexicographique ne suffit pas pour prouver que l'expression mathématique est syntaxiquement correcte.

Remarquons toutefois que l'analyse lexicographique et l'analyse syntaxique ne sont pas nécessairement globalement séquentielles dans le temps. Il se peut que la liste des atomes ne soit pas construite avant de commencer l'analyse syntaxique.

3.1.3 Le rôle de l'analyseur syntaxique

Le but de l'analyse syntaxique est de déterminer si une chaîne peut être engendrée par la grammaire que nous avons définie. Il s'agit donc, étant donné une chaîne ω, de retrouver la suite de dérivations qui, de l'axiome S, conduit à la chaîne étudiée. Si nous pratiquons ceci à la main pour de petits exemples, nous pouvons utiliser la technique classique dite « essais-erreurs »[15]. Nous verrons que cette suite se présente bien plus clairement sous la forme d'un arbre (cfr. 3.1.4. L'arbre de décomposition syntaxique, page 52).

L'analyseur syntaxique obtient une chaîne d'unités lexicales de l'analyseur lexicographique. Son but est d'obtenir l'axiome S. Pour cela, il teste si l'enchaînement des unités lexicales est syntaxiquement correct en leur appliquant des règles syntaxiques de notre grammaire. Il devra signaler chaque erreur de façon intelligible (envoi d'un message à l'écran, erreur mise en surbrillance, ...).

Nous savons qu'une expression mathématique peut contenir des erreurs à différents niveaux, par exemple,

- lexicographiques, comme les caractères non valides par rapport à notre alphabet;
- syntaxiques, comme une expression arithmétique mal parenthésée
- sémantiques, comme un opérateur appliqué à une opérande incompatible

Le gestionnaire d'erreurs d'un analyseur syntaxique a des buts simples à énoncer :

- il doit indiquer la présence d'erreurs de façon claire et précise;
- il doit traiter chaque erreur suffisamment rapidement pour pouvoir détecter les erreurs suivantes;
- il ne doit pas ralentir de façon significative le travail de l'utilisateur.

[15] Cette méthode consiste à tenter de deviner à partir d'une chaîne la suite des dérivations qui ont permis de l'engendrer.

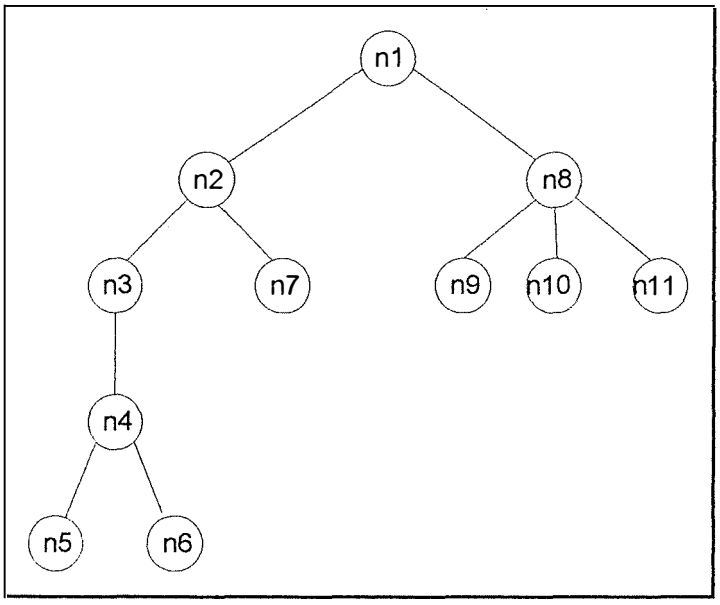
3.1.4 L'arbre de décomposition syntaxique

3.1.4.1 Les définitions

La structure d'arbre est fondamentale en informatique. *« Elle permet de représenter de façon structurée et très efficace des notions qui se présentent sous forme d'une chaîne de caractères » [ARMI86].* Ainsi, l'analyse syntaxique fait partie des nombreuses situations où l'*« on transforme une entité, qui se présente sous une forme plate et difficile à manipuler, en une forme structurée adaptée à un traitement efficace »[ARMI86].*

Les structures d'arbres possèdent un double intérêt : d'une part, *« les données qui interviennent dans de nombreux problèmes sont naturellement structurées en arbres (hiérarchies d'objets, choix et décisions, arbres syntaxiques, etc.) » [ARMI86]*; d'autre part, *« elles permettent de représenter efficacement des ensembles d'objets ou des applications, nous parlerons dans ce cas d'arbres de recherche (search trees) » [ARMI86].*

Un arbre est soit un arbre atomique (une *feuille*), soit un *noeud* et une suite de sous-arbres. Graphiquement, un arbre peut être représenté comme suit,



Le noeud n_1 est la *racine* de l'arbre; n_5, n_6, n_7, n_9, n_{10}, n_{11} sont les *feuilles*; n_1, n_2, n_3, n_4, n_8 sont les *noeuds internes*. Plus généralement, l'ensemble des *noeuds* est constitué des noeuds internes et des feuilles. Contrairement à la botanique, on dessine les arbres avec la racine en haut et les feuilles vers le bas en informatique. Si une branche relie un noeud n_i à un noeud n_j plus bas, nous dirons que n_i est un *ancêtre* de n_j. Une propriété fondamentale d'un arbre est qu'un noeud n'a qu'un seul père. Les *arbres binaires* sont des arbres tels que les noeuds ont au plus deux fils. La *hauteur*, appelée aussi *profondeur* d'un noeud, est la longueur du chemin qui le joint à la racine, ainsi la racine est elle-même de hauteur 0, ses fils de hauteur 1 et les autres noeuds de hauteur supérieure à 1. Le *degré* d'un noeud est le nombre de ses enfants.

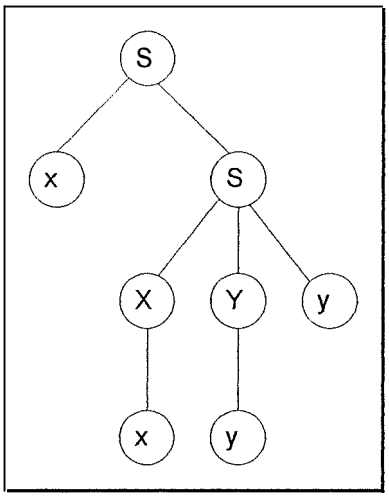
En tant que structure, un arbre est défini comme,

- un arbre vide;
- ou bien un noeud racine qui possède une valeur et qui est lié à 0, 1 ou plusieurs sous-arbres éventuellement vides.

Si l'ordre entre les sous-arbres enfants est pris en compte, nous parlerons d'arbre ordonné (à ne pas confondre avec un arbre trié).

La représentation arborescente est un moyen idéal pour vérifier la syntaxe d'une expression mathématique. En effet, d'une part, les règles syntaxiques permettent de passer d'une catégorie syntaxique à une autre, créant des liens hiérarchiques entre chaque catégorie syntaxique. D'autre part, nous allons essayer de construire un arbre de décomposition syntaxique correspondant à la structure de l'expression mathématique introduite par l'utilisateur (suite des dérivations de l'expression).

Si nous reprenons l'exemple de la page 48, une structure de la chaîne xxyy peut être représentée simplement par l'arbre orienté suivant :



Pour une même chaîne donnée, du langage engendré par une grammaire, l'arbre de décomposition syntaxique n'est pas nécessairement unique. L'existence de plusieurs arbres de décomposition syntaxique pour une même chaîne signifie qu'il existe plusieurs interprétations possibles pour celle-ci. Nous disons alors que la grammaire est « ambiguë ». En Pascal, c'est le cas pour l'imbrication des « *if then* » et « *if then else* ». Des informations supplémentaires dans le manuel de référence du langage permettent de lever l'ambiguïté et d'associer un arbre unique à tout programme Pascal. Ceci permet alors de donner une interprétation unique.

3.1.4.2 La construction de l'arbre de décomposition syntaxique

L'arbre de décomposition syntaxique est construit à partir de la suite de dérivations de la chaîne ω. Chaque noeud est « valué » par un symbole du vocabulaire $V = V_T \cup V_N \cup \Lambda$. Plus précisément, toute feuille est « valuée » par le symbole terminal dont elle est l'occurrence dans ω. Un noeud interne est « valué » par le symbole non terminal dont il est l'occurrence dans la suite de dérivations de la chaîne ω.

A tout noeud interne, nous pouvons associer *« l'intervalle de la chaîne ω qui en dérive, qui est une proposition[16], et que nous appelons une sous-proposition de la chaîne ω » [PAIR64].*

L'arbre de décomposition syntaxique représente *« l'arbre des sous-propositions et des occurrences de symboles terminaux formant la chaîne ω, il est muni de la relation R :*

$\alpha R \beta \Leftrightarrow \beta \subset \alpha$, *si* β *est une sous-proposition*
$\alpha R \beta \Leftrightarrow \beta \in \alpha$, *si* β *est une occurrence terminale » [PAIR64]*

Nous appellerons *phrases* ϖ les propositions qui dérivent d'un axiome fixé S dans l'ensemble V_N. Tandis que les sous-propositions d'une phrase seront appelées *sous-phrases* (bien qu'elles ne soient pas des phrases).

Remarque : Pour un noeud interne d'étiquette S, la chaîne ρ obtenue en lisant de gauche à droite les étiquettes de ses fils est tel que $S \rightarrow \rho$ est une règle. La chaîne ω dont nous faisons l'analyse est constituée des étiquettes des feuilles lues de gauche à droite.

3.1.4.3 Un exemple de construction d'arbre de décomposition syntaxique : Les expressions arithmétiques

Les chaînes engendrées par la grammaire suivante sont toutes les expressions arithmétiques que nous pouvons écrire avec les opérateurs * et +. On les appelle parfois *« expressions arithmétiques infixes » [LERO94]*. Nous les interprétons en disant que * est prioritaire vis à vis de +.

Soient le vocabulaire terminal V_T = {a, b, +, *, }
le vocabulaire non terminal V_N = {Expression, Terme, Facteur }
les règles syntaxiques suivantes constituant la grammaire

< Expression > ::= < Expression > + < Terme >
| < Terme >
< Terme > ::= < Terme > * < Facteur > | < Facteur >
< Facteur > ::= a | b

Dans cette grammaire, nous constatons que toute expression est somme de termes et que tout terme est produit de facteurs. Chaque facteur est réduit à la variable *a* ou bien à la variable *b*.

L'axiome S est < Expression > et « a *b + a » est une phrase de la grammaire qui peut être construite grâce à la suite de dérivations,

φ_1 = < Expression >
φ_2 = < Expression > + < Terme >
φ_3 = < Expression > + < Facteur >
φ_4 = < Expression > + a

[16] Nous appelons « proposition » d'une grammaire G, toute chaîne ω de symboles terminaux qui est une dérivation d'un élément S quelconque de V_N. La grammaire G est construite sur un vocabulaire terminal V_T et décrite grâce à un vocabulaire non terminal V_N [PAIR64].

$\varphi_5 = <\text{Terme}> + a$
$\varphi_6 = <\text{Terme}> * <\text{Facteur}> + a$
$\varphi_7 = <\text{Terme}> * b + a$
$\varphi_8 = <\text{Facteur}> * b + a$
$\varphi_9 = a * b + a = \varpi$

La convention usuelle de priorité de l'opération * sur +, explique que nous commençons à engendrer des sommes de termes avant de décomposer les termes en produits de facteurs. En règle générale, pour des opérateurs de priorités quelconques, nous commençons à engendrer les symboles d'opérations ayant la plus faible priorité pour terminer par ceux qui correspondent aux plus fortes.

Pour la phrase ϖ les figures 3.1, 3.2 et 3.3 représentent respectivement l'arbre de décomposition syntaxique A, l'arbre des sous-phrases $\tilde{A}$, l'arbre des sous-phrases et des occurrences terminales A* : chaque sous-phrase est représentée par la chaîne de ses éléments, mise entre parenthèses[17].

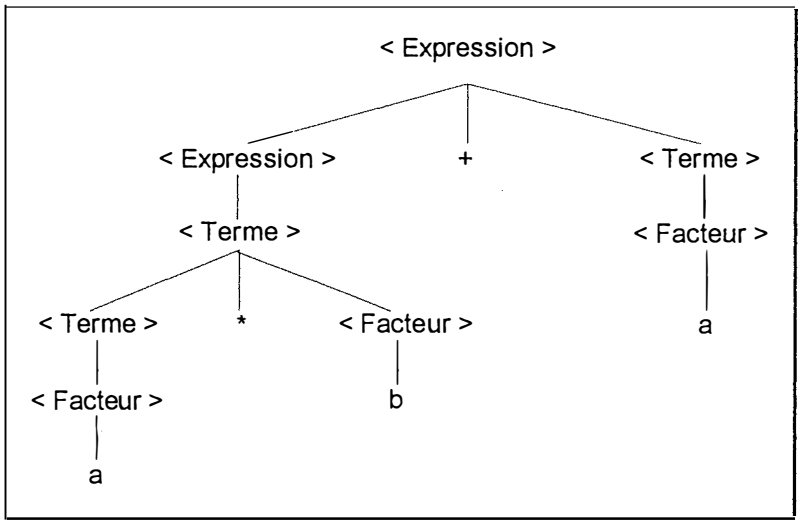


Figure 3.1 L'arbre de décomposition syntaxique A

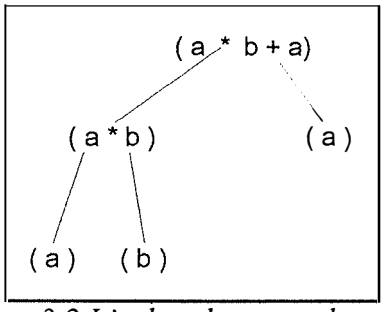


Figure 3.2 L'arbre des sous-phrases $\tilde{A}$

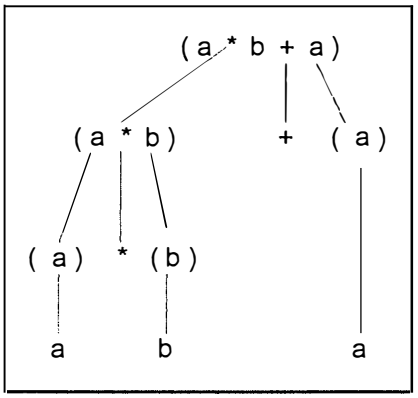


*Figure 3.3 L'arbre des sous-phrases et des occurrences terminales A**

[17] Nous avons emprunté la terminologie de l'article de [PAIR64].

Remarque : Il est très facile d'ajouter de nouvelles règles syntaxiques. Il faudra faire intervenir de nouvelles catégories syntaxiques si nous introduisons de nouvelles priorités. Si nous souhaitons introduire des soustractions et des divisions, il n'est pas nécessaire d'introduire de nouveaux éléments dans V_N. En effet, ces deux opérations ont respectivement la même priorité que l'addition et la multiplication.

3.1.5 Les méthodes d'analyse syntaxique

Les deux grands types d'analyseur syntaxique sont,

- d'une part, les analyseurs descendants (« *Top down* ») qui construisent les arbres d'analyse de haut en bas;
- d'autre part, les analyseurs ascendants (« *Bottom up* ») qui partent des feuilles et remontent vers la racine.

Ces deux méthodes d'analyse *« ne tiennent pas compte des ambiguïtés d'une grammaire puisqu'elles ne recherchent qu'une seule structure pour une chaîne donnée » [BRAS65].* Une troisième méthode d'analyse, appelée « analyse multiple », permet de les obtenir toutes.

La méthode que nous avons choisie d'adopter pour l'analyseur syntaxique est la méthode ascendante. Selon [BRAS65], *« les algorithmes d'analyse ascendante sont plus souvent compliqués que ceux de l'analyse descendante; cependant, ils s'appliquent à un plus grand nombre de grammaires ».* De plus, l'algorithme que nous utiliserons pour l'analyse des expressions arithmétiques infixes engendrées par notre grammaire (cfr. 5.2.1 La définition de notre grammaire) n'admet pas d'analyse syntaxique descendante simple. En effet, d'une part, l'analyse descendante ne donne pas de résultat lorsque la grammaire est ce que nous appelons *récursive à gauche*, c'est-à-dire lorsqu'il existe une suite de dérivations partant d'un axiome S de V_N et conduisant à une chaîne ω qui débute par S (par exemple, la règle 3 de notre exemple page 48). D'autre part, cette méthode est très coûteuse en temps si nous analysons une chaîne assez longue. En effet, nous effectuons tous les essais successifs des règles et nous pouvons parfois nous rendre compte, après avoir terminé l'analyse, que la première règle que nous avons appliquée n'est pas la bonne. Dès lors, il faut tout recommencer avec une autre règle et éventuellement répéter le mécanisme plusieurs fois. La complexité de l'algorithme est ainsi une fonction exponentielle de la longueur de la chaîne à analyser.

Comme nous l'avions expliqué précédemment, étant donnés une grammaire G et une chaîne ω, il faut construire la suite des dérivations qui, de l'axiome S, conduit à la chaîne ω,

$$S \rightarrow \rho_1 \rightarrow \rho_2 \ldots \rho_{n-1} \rightarrow \rho_n = \omega$$

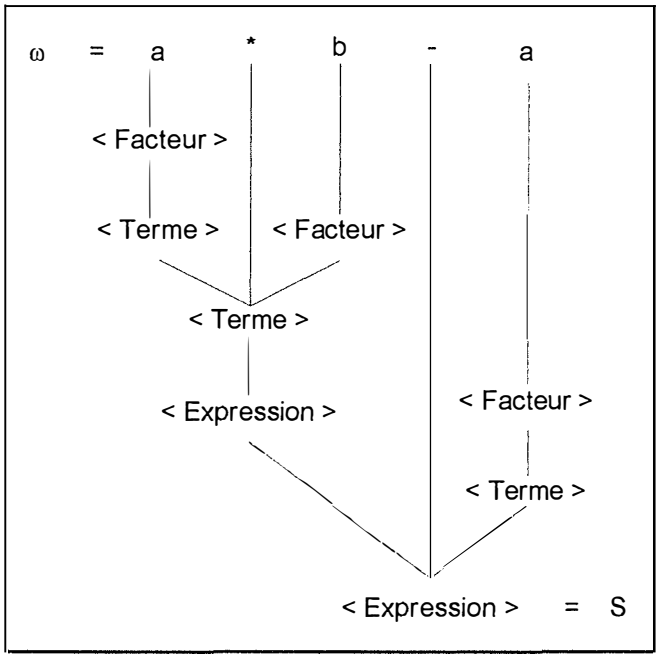
L'analyse ascendante consiste à commencer par deviner ρ_{n-1} à partir de ω puis de remonter à ρ_{n-2} et successivement jusqu'à l'axiome S. L'analyse descendante opère en sens inverse (il faut démarrer de l'axiome et tenter de retrouver ρ_1, puis ρ_2 jusqu'à obtenir $\rho_n = \omega$).

Dans l'analyse ascendante, nous effectuons ce qu'on appelle des *réductions* car il s'agit de remplacer un membre de droite d'une règle syntaxique par le membre de gauche correspondant. Celui-ci est en général plus court.

En reprenant la grammaire décrite au point 3.1.4.3 et en transformant une des règles syntaxiques de la grammaire, < Expression > ::= < Expression > + < Terme > | < ***Expression*** > - ***< Terme >*** | < Terme >, nous allons présenter ici ce qu'on appelle *l'automate* qui effectue l'analyse syntaxique de la grammaire, récursive à gauche, des expressions infixes. Nous allons lire la chaîne ω (a * b - a) de gauche à droite. Remarquez que la soustraction n'est pas associative. Ainsi la technique de l'analyse descendante ne peut pas être appliquée dans ce cas-ci. Nous pouvons effectuer les réductions suivantes dès qu'elles sont possibles,

- Réduire *a* en *Facteur* quelle que soit sa position
- Réduire *b* en *Facteur* quelle que soit sa position
- Réduire *Facteur* en *Terme* s'il n'est pas précédé de *
- Réduire *Terme* * *Facteur* en *Terme* quelle que soit sa position
- Réduire *Terme* en *Expression* s'il n'est pas précédé de + et s'il n'est pas suivi de *
- (Réduire *Expression* + *Terme* en *Expression* s'il n'est pas suivi de *)[18]
- Réduire *Expression* - *Terme* en *Expression* s'il n'est pas suivi de *

Nous obtenons le même arbre de décomposition syntaxique (en commençant par les feuilles de l'arbre) que celui de la figure 3.1.



Remarque : Si, à un moment de la construction de cet arbre, il n'avait plus été possible de réaliser une réduction et que nous nous soyons retrouvés en présence d'une *forêt d'arbres*[19], cela aurait signifié que la chaîne ω ne respectait pas notre grammaire G.

[18] Cette réduction n'est pas nécessaire dans cet exemple.

[19] Il s'agit d'un graphe sans cycle dont les composants simplement connexes sont des arbres.

3.2 L'analyseur sémantique

L'analyse sémantique est la partie essentielle de l'analyse d'un texte introduit par l'utilisateur. Elle devra fournir une représentation symbolique du sens de l'équation du problème. L'analyseur syntaxique, décrit précédemment, fournira des structures syntaxiques (arbres de décomposition syntaxique) utilisées pour l'analyse sémantique. La phase d'analyse sémantique contrôle si l'expression mathématique contient des erreurs sémantiques et collecte des informations destinées au gestionnaire d'erreurs.

3.2.1 La vérification sémantique au niveau de la mise en équation du problème et au niveau de la résolution de l'équation

Nous allons contrôler la sémantique en réalisant deux types de comparaison relatifs à la représentation de l'équation :

- équation sous forme canonique ax + b = 0;
- équation sous forme d'arbre de décomposition syntaxique.

Nous devons distinguer la vérification sémantique qui se fait au niveau de la mise en équation du problème et celle au niveau de la résolution de l'équation.

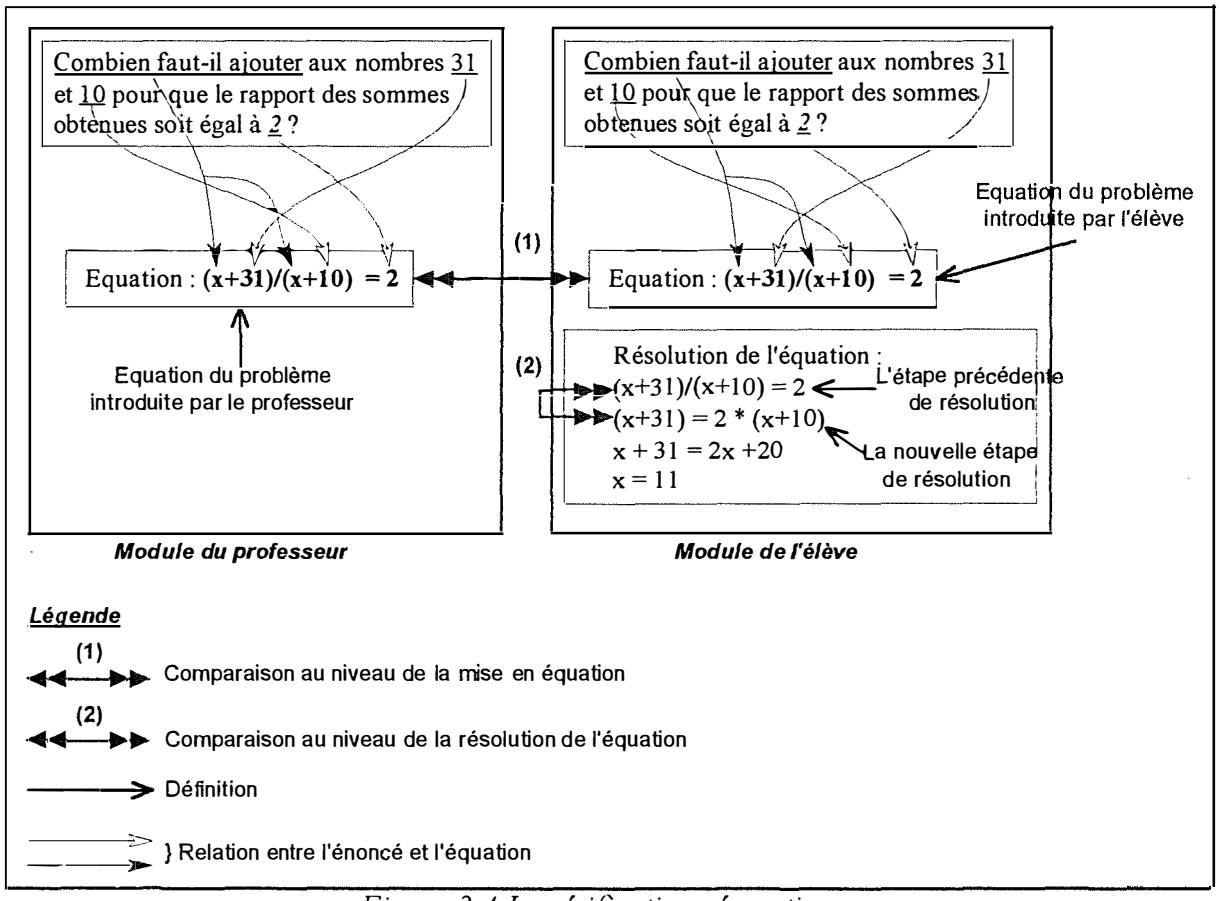


Figure 3.4 La vérification sémantique

Pour résoudre l'équation du problème, l'élève va effectuer plusieurs simplifications et/ou regroupements d'éléments de l'équation. Chacune de ces opérations entraînera l'écriture d'une nouvelle expression mathématique se rapprochant, normalement[20], de la solution de l'équation. Nous devons vérifier si cette nouvelle équation est bien issue de l'étape précédente de résolution de l'équation.

Au niveau de la mise en équation du problème, nous effectuons des comparaisons entre l'équation introduite par le professeur et celle trouvée par l'élève. Préalablement, nous effectuons une comparaison de leur équation canonique. Ensuite, si les équations canoniques sont identiques, nous vérifions si l'équation est bien en rapport avec l'énoncé du problème. Pour le savoir, il faut comparer les relations établies par l'élève, entre l'énoncé et l'équation, avec celles réalisées par le professeur (les zones de sélection minimale et maximale). Finalement, si les correspondances entre l'énoncé et l'équation sont satisfaisantes, nous réalisons la comparaison des arbres de décomposition syntaxique.

A partir de maintenant, nous appellerons l'équation du problème introduite par le professeur et l'étape précédente de la résolution (cfr. figure 3.4), « l'expression précédente ». Tandis que l'équation du problème introduite par l'élève et la nouvelle étape de résolution (cfr. figure 3.4), s'appelleront « l'expression en cours d'analyse ».

3.2.2 La comparaison des équations canoniques

Les expressions ont été mises sous la forme canonique $ax + b = 0$. Nous allons montrer que si deux équations canoniques sont identiques, cela ne signifie pas nécessairement qu'elles sont sémantiquement équivalentes.

Une analyse sémantique rapide serait de comparer l'équation canonique issue de l'expression précédente et celle issue de l'expression en cours d'analyse.

Si elles sont différentes, l'analyseur sémantique sait alors que l'expression en cours d'analyse est sémantiquement incorrecte. Il envoie un message d'erreur au gestionnaire d'erreurs. Il est possible d'affiner le message d'erreur en fonction du système (il reconnaît l'expression comme incorrecte ou bien il ne peut pas la juger).

Si les équations canoniques sont identiques, nous ne pouvons pas encore déterminer si elles sont sémantiquement équivalentes.

Soient l'expression précédente « 150x + (12-x)*100 = 1450 »,
l'expression en cours d'analyse « 100x + (10-x)*50 = 750 »,

Ces deux expressions ont la même équation canonique, à savoir, 50x -250 = 0. Mais, sont-elles vraiment sémantiquement équivalentes? L'analyseur sémantique va alors effectuer un contrôle basé sur les arbres de décomposition syntaxique créés par l'analyseur syntaxique pour le déterminer.

[20] Une méthode, pour vérifier si l'élève ne s'éloigne pas dans la résolution de l'équation, est de vérifier si le nombre de noeuds de l'arbre de décomposition syntaxique ne croit pas démesurément.

3.2.3 La comparaison des arbres de décomposition syntaxique

Si l'analyse ascendante ne recherche qu'une seule structure pour une chaîne donnée, il existe cependant une multitude de représentations équivalentes pour une même expression algébrique,

- Chaque élément algébrique est décomposable en expressions plus simples (factorisation) ou plus complexes (distributivité). La notion de « complexité » est subjective car elle dépend de notre capacité à pouvoir apprécier si une expression est complexe ou non.

  Exemple : l'expression algébrique (ac + bc) + (ac - dc) est sémantiquement équivalente à l'expression c* (2a + b - d).

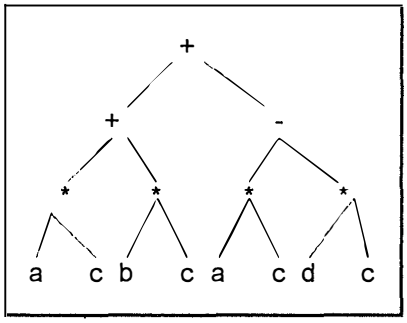


Figure 3.5 L'arbre de décomposition syntaxique de (ac + bc) + (ac - dc)

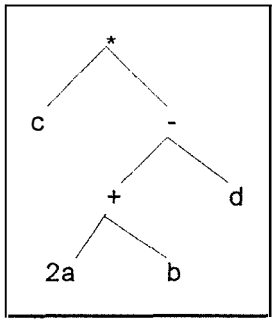


Figure 3.6 L'arbre de décomposition syntaxique de c (2a + b - d)*

Ces deux arbres de décomposition syntaxique sont sémantiquement équivalents même si leur structure est différente.

- L'addition et la multiplication sont ***commutatives***. Ce qui signifie qu'il est possible de permuter leurs opérandes. Si une expression algébrique est composée d'un nombre *n* d'opérateurs + (ou *), alors il existe *(n+1)!* expressions algébriques sémantiquement équivalentes et par conséquent, *(n+1)!* manières de les représenter sous forme d'arbres.

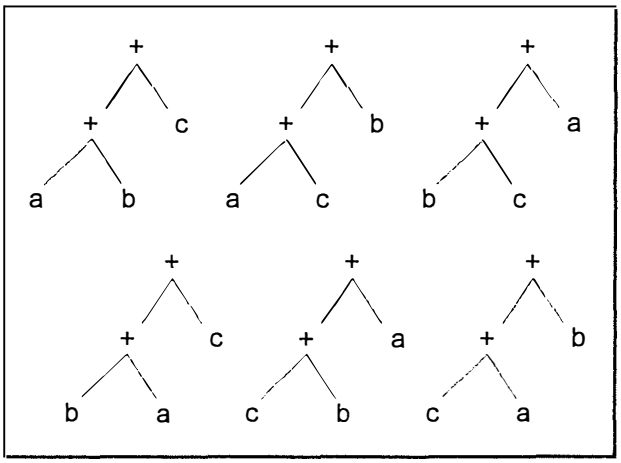


Figure 3.7 Les arbres de décomposition syntaxique sémantiquement équivalents pour n = 2, l'opérateur + et les opérandes a, b, c

Notre objectif est de pouvoir trouver une modélisation unique pour des expressions multiples, mais sémantiquement équivalentes. Nous souhaitons « standardiser » les arbres de décomposition syntaxique.

3.2.3.1 La normalisation des arbres

Les opérateurs qui interviendront dans nos expressions sont les suivants : * / + - ^
Quelles types d'opération pouvons-nous effectuer?

OPERATIONS	*	/	+	-	^
♦ Effectuer	x	x	x	x	x
♦ Réduire au même dénominateur	x				
♦ Neutraliser le coefficient de l'inconnue	x	x			
♦ Distribuer	x				x
♦ Simplifier		x			

La soustraction et la division ne sont pas commutatives. Il n'est donc pas possible de permuter les opérandes de ces opérations. Par conséquent, pour atteindre notre objectif (la modélisation unique pour des expressions équivalentes sémantiquement), nous allons transformer l'arbre de décomposition syntaxique en un arbre composé uniquement d'opérateurs « + » et « * ».

A. Transformons les soustractions et les divisions respectivement en additions et en multiplications

Nous allons donc changer les soustractions en additions. Il suffit de transformer le sous-arbre droit, dont la racine est l'opérateur « - », en son opposé.

- L'opposé d'un nombre (ou d'un terme contenant l'inconnue) s'obtient en multipliant celui-ci (ou le coefficient de l'inconnue) par *-1;*
- L'opposé d'une somme (de « + » ou de « - ») s'obtient en faisant la somme des opposés;
- L'opposé d'un produit (de « * » ou de « / ») est le produit de l'opposé d'un facteur par l'autre facteur ou les autres;

Ensuite, nous remplacerons les divisions par des multiplications. Il faut transformer le sous-arbre droit, dont la racine est l'opérateur « / », en son inverse.

- L'inverse d'un nombre n est égal à $\frac{1}{n}$;
- L'inverse d'un terme contenant l'inconnue s'obtient en inversant son coefficient et en multipliant la puissance de l'inconnue par -1;
- L'inverse d'une somme (de « + » ou de « - ») s'obtient en élevant la somme à la puissance -1;
- L'inverse d'un produit s'obtient en faisant le produit des inverses;
- L'inverse d'un quotient s'obtient en faisant le quotient des inverses;

En ce qui concerne l'opérateur « ^ » ne s'appliquant qu'à des nombres (cfr. 5.2.1 La définition de notre grammaire), nous effectuerons toujours la puissance.

Nous obtenons finalement un arbre algébriquement identique à l'arbre de décomposition syntaxique. Nous l'appellerons dorénavant *l'arbre +/**.

B. Les propriétés de l'addition et de la multiplication

Propriétés « + »
♦ Associativité
♦ Commutativité
♦ Neutre : 0
♦ Opposé : -a

Propriétés « * »
♦ Associativité
♦ Commutativité
♦ Neutre : 1
♦ Inverse : a^{-1}
♦ Distributive par rapport à « + »
♦ Absorbant : 0

Les propriétés de l'élément ***neutre*** (figure 3.8), pour l'addition et de la multiplication, et de l'élément ***absorbant*** (figure 3.9) pour la multiplication, peuvent permettre de réduire notre arbre +/*.

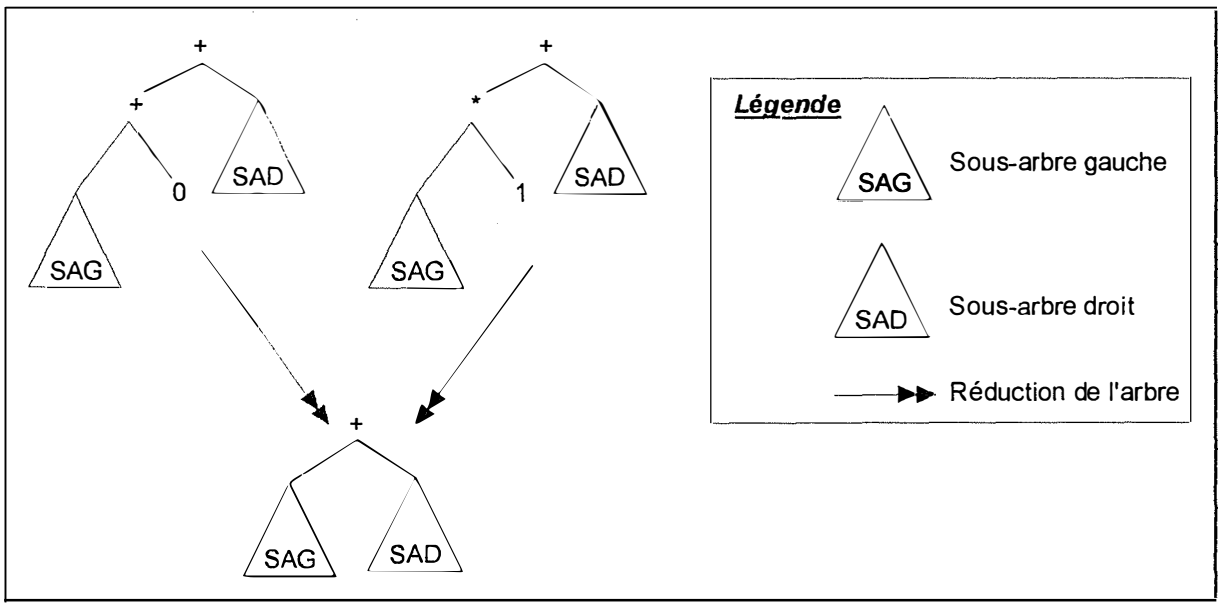


Figure 3.8 La propriété du neutre de l'addition et de la multiplication

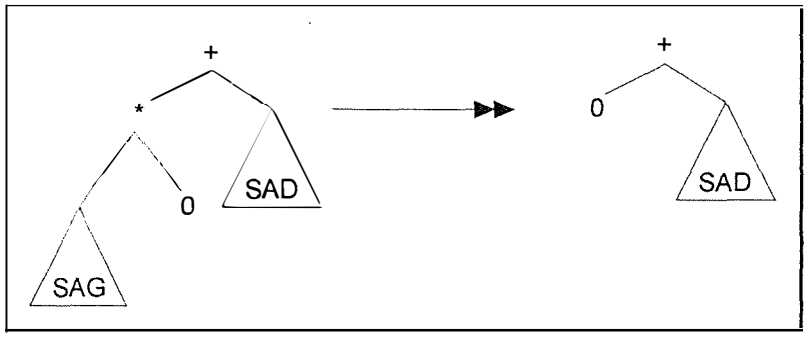


Figure 3.9 La propriété de l'absorbant pour la multiplication

La propriété de ***l'associativité*** nous permet de transformer un arbre contenant un nombre *n* de noeuds contigus constitué du même opérateur, en un arbre (n+1)-aire regroupant leurs opérandes sous le noeud étiqueté « Somme » pour l'opérateur « + » et « Produit » pour l'opérateur « * ». Nous obtenons un nouvel arbre algébriquement identique à l'arbre initial.

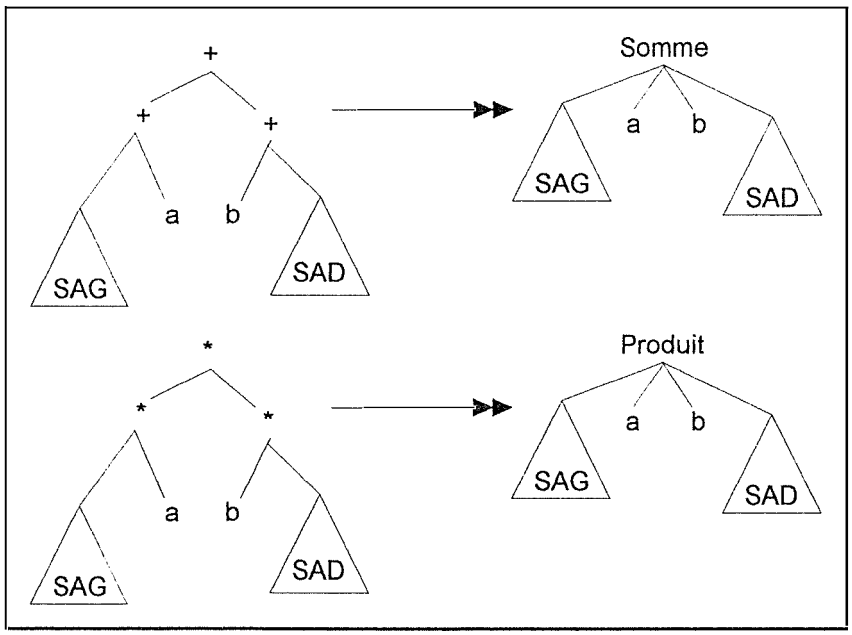
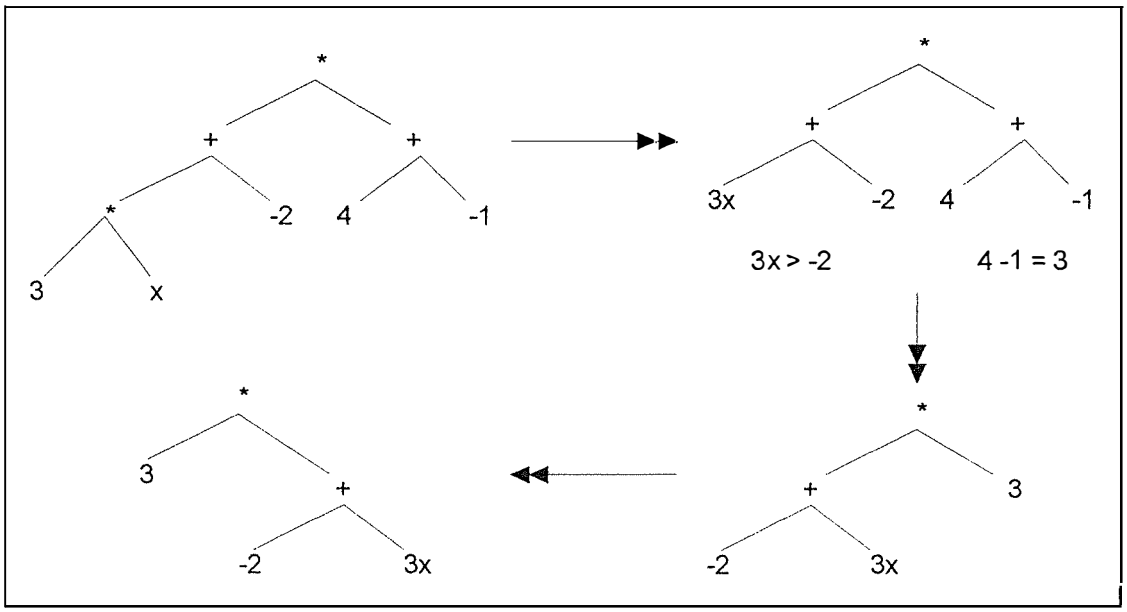


Figure 3.10 La propriété de l'associativité pour la multiplication et l'addition

La figure 3.7 montre qu'il est possible de construire plusieurs arbres sémantiquement équivalents en intervertissant les sous-arbres gauches et droits du premier arbre. La propriété de ***commutativité*** nous permet également d'ordonner les feuilles, constituées soit d'un nombre ou d'un terme contenant l'inconnue, de l'arbre +/*. Tout en parcourant l'arbre +/* de manière infixée[21], nous regardons le sous-arbre. S'il est composé uniquement de valeurs numériques, alors nous calculons le résultat de l'opération. Sinon, nous devons ordonner le polynôme en comparant les chaînes de caractères.

Feuilles alpha-numérique > Feuille numérique Sous-arbre « polynomial » > Sous-arbre numérique



[21] Nous parcourons le sous-arbre gauche, puis la racine et finalement le sous-arbre droit.

3.2.3.2 La comparaison des arbres +/*

Nous venons de normaliser l'arbre de décomposition syntaxique en le transformant en une représentation unique d'arbre +/* simplifié. « Simplifié » puisque nous avons réalisé des regroupements et des simplifications d'éléments similaires ainsi qu'appliquer les propriétés de l'addition et de la multiplication. L'arbre de décomposition est sémantiquement équivalent à l'arbre +/* simplifié.

La dernière étape est de comparer l'arbre +/* de l'expression précédente avec celui de l'arbre de l'expression en cours d'analyse. Il faut donc rechercher s'il y a un sous-arbre équivalent. Nous essayons de faire correspondre des éléments de l'arbre +/* de l'expression en cours d'analyse avec ceux de l'arbre de l'expression précédente, ayant une racine commune. Si, pour un sous-arbre de l'arbre +/* de l'expression en cours d'analyse, nous ne trouvons pas de correspondance, nous demandons alors à l'élève d'où provient la chaîne de caractères issue de ce sous-arbre.

Remarque : En permettant à l'élève de faire du « Copier-Coller »[22], nous pouvons réduire notre analyse sémantique puisque la partie qu'il a sélectionnée dans l'expression précédente et ajoutée dans l'expression en cours d'analyse, a forcément la même signification sémantique. Il faut néanmoins que cette partie s'insère proprement dans l'expression (pas d'erreur syntaxique). Dans les exemples qui suivent, la sélection de l'élève sera mise en caractères gras.

EXEMPLES

N°	EXPRESSION PRECEDENTE	EXPRESSION EN COURS D'ANALYSE
1	**150* x** + (12-x)*100 = **1450**	**150* x** + + 1200 -100*x = **1450**
2	**150* x + 1200 -100*x = 1450**	**150* x + -100*x = 1450 -1200**

Ces deux exemples illustrent les avantages et les inconvénients du « Copier-Coller ».

Les avantages de cet outil (Cas N°2) sont tels qu'ils permettent à l'élève de reprendre un morceau de l'expression précédente pour l'introduire dans sa nouvelle équation (expression en cours d'analyse). Il établit un lien entre les deux expressions. De plus, lorsqu'il sera habitué à la manipulation de la fonction « Copier-Coller », il pourra même gagner du temps dans la rédaction de cette nouvelle équation. Pour notre didacticiel, cela constitue également un gain de temps au niveau de l'analyse sémantique.

Cependant, l'élève doit rester vigilant lorsqu'il emploie cette fonction. Dans le premier cas, nous voyons qu'il a oublié avoir déjà recopié l'opérateur « + » et lorsqu'il a effectué la distributivité, il a ajouté une fois de plus l'opérateur. Ce qui entraîne une erreur syntaxique dans son expression (une succession d'opérateurs « + » ne sera pas admise par notre grammaire).

[22] L'élève a la possibilité de sélectionner une partie de l'expression précédente, dans l'éditeur d'équations, ensuite de la copier, puis de l'introduire (en la collant) dans l'expression qu'il est en train d'écrire (nouvelle étape dans la résolution de l'équation).

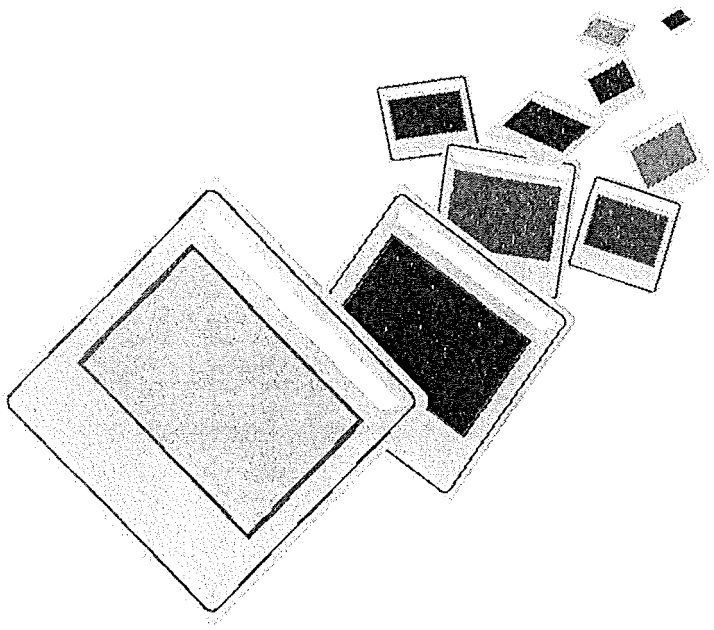
Conclusion

Nous venons de parcourir les différents concepts et stratégies sur lesquels nous avons basé la conception de notre didacticiel. A travers ce chapitre, nous avons pu mieux percevoir les rôles des différents types d'analyse (lexicographique, syntaxique et sémantique) que nous allons devoir implémenter dans notre application.

Nous nous sommes donnés des outils et des méthodes qui vont nous permettre de valider la syntaxe et la sémantique d'une expression introduite par l'utilisateur. Pour ce faire, nous avons eu besoin, entre autres, de mécanismes permettant de transformer cette expression, de sa représentation externe (chaîne de caractères), vers une représentation interne (arbre binaire).

Chapitre 4

Une proposition pour l'interface



Introduction

Dans ce chapitre, nous allons vous présenter l'interface de notre didacticiel. Comme vous le savez, notre logiciel d'apprentissage est composé de deux grands modules, le module du professeur et le module de l'élève. Pour ces deux modules, les diverses fenêtres sont fort semblables, aussi nous ne définirons la présentation que d'un seul module. Cependant, nous ne manquerons pas de signaler les différences.

Dans un premier temps, du graphe d'enchaînement des fonctions sémantiques, nous avons dégagé les unités de présentation correspondant à une sous-tâche. Des procédures internes au système, comme l'analyse syntaxique et l'analyse sémantique, ne seront pas présentes dans les graphes d'enchaînement. Bien que le système envoie un message d'erreur à l'utilisateur si le résultat de ces analyses est négatif, nous n'avons pas voulu surcharger nos graphes. Les messages d'erreur entre l'utilisateur et le système, qui sont très nombreux, ne figureront pas non plus dans nos graphes. Ensuite, nous avons choisi les attributs de dialogue.

L'étape suivante consiste à identifier les fenêtres.

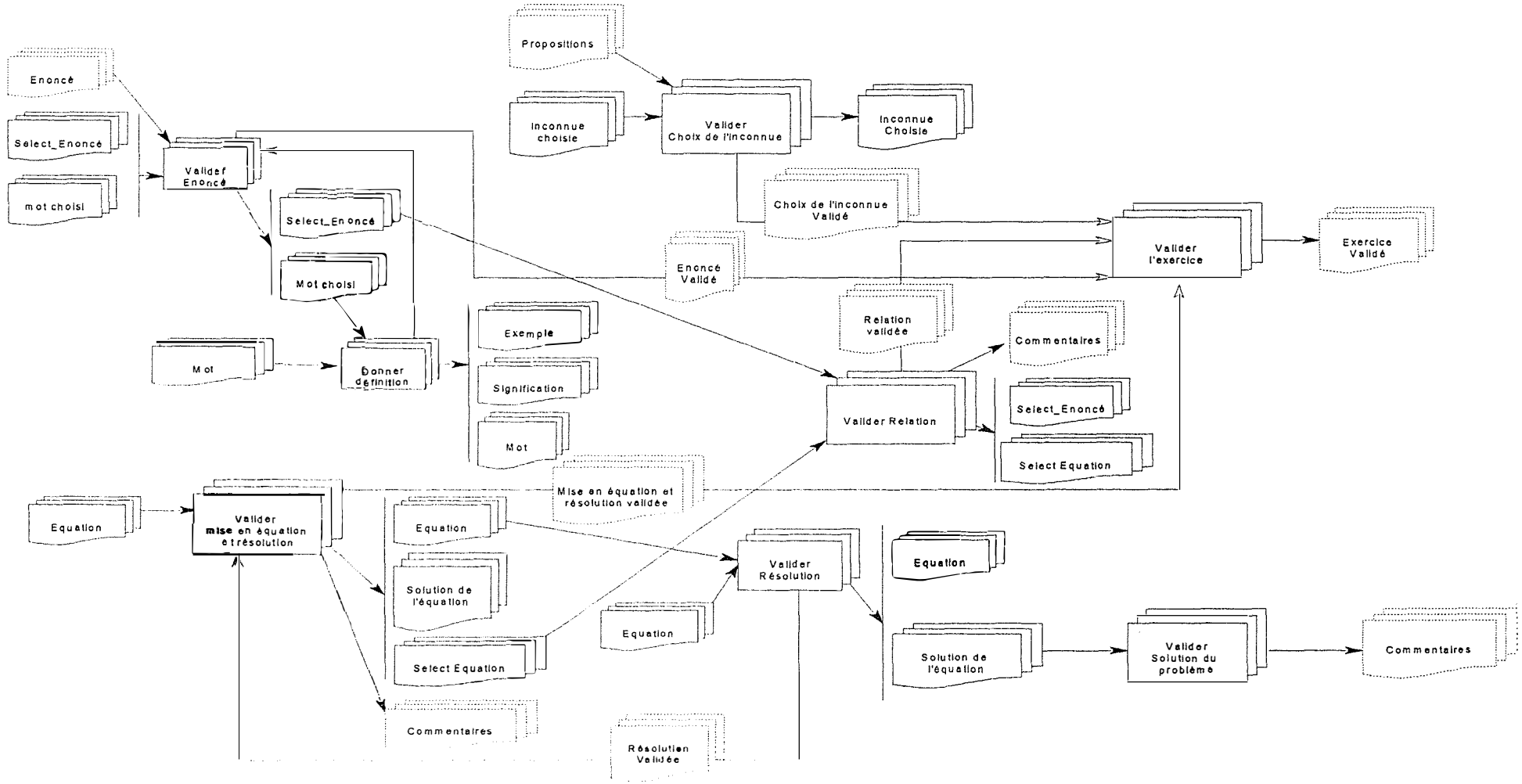
Pour chaque fenêtre identifiée, nous faisons correspondre une fenêtre physique tandis qu'au sein de chaque fenêtre, nous ferons correspondre un objet interactif abstrait (O.I.A.) pour chacune des informations en entrée ou en sortie ainsi que pour chacune des fonctions.

Pour chaque O.I.A., nous allons faire correspondre les objets interactifs concrets (O.I.C.) propres à notre environnement de travail, Microsoft Windows.

Finalement, nous décrivons l'aspect des fenêtres que nous avons créées, tout en essayant de respecter les règles de placement des O.I.C. au sein des fenêtres. Les principales règles sont issues de [VAND92] et elles nous ont permis de définir nos écrans.

4.1 L'identification des unités de présentation

4.1.1 La rédaction du graphe d'enchaînement des fonctions sémantiques pour le module de l'élève



4.1.2 La rédaction du graphe d'enchaînement des fonctions sémantiques pour le module du professeur

A partir du graphe d'enchaînement des fonctions sémantiques du module du professeur (Cfr. page 70), nous dégageons deux unités de présentation. Chacune de ces unités de présentation correspond à une sous-tâche, *Validation du mot de passe* (UP1) et *Validation des exercices* (UP2). Ceci est inspiré par la règle méthodologique : « *une unité de présentation par sous-tâche* ». Chaque unité de présentation comprend une fenêtre principale à partir de laquelle les fenêtres de l'unité sont enchaînées. Du graphe d'enchaînement pour le module de l'élève, nous n'avons pu dégager qu'une seule unité de présentation. Aussi, nous nous intéresserons davantage à l'interface du module du professeur.

Le choix des attributs de dialogue pour le module du professeur

Les quatre attributs de dialogue caractérisant le dialogue de l'interface sont,

A. Le contrôle du dialogue

Les fonctions *Valider Mot de passe, Valider Enoncé, Ajouter définition, Valider Relation, Valider Solution du problème, Valider Choix de l'inconnue,* sont globalement externes pour la tâche, c'est-à-dire à l'initiative de l'utilisateur. Certaines fonctions comme *Valider Mise en équation et résolution* le sont partiellement puisque le système, dans ce cas-ci, calcule la solution de l'équation.

La fonction *Valider l'exercice* est cependant interne puisque le système doit vérifier si tout a été introduit par l'utilisateur.

B. Le mode de dialogue

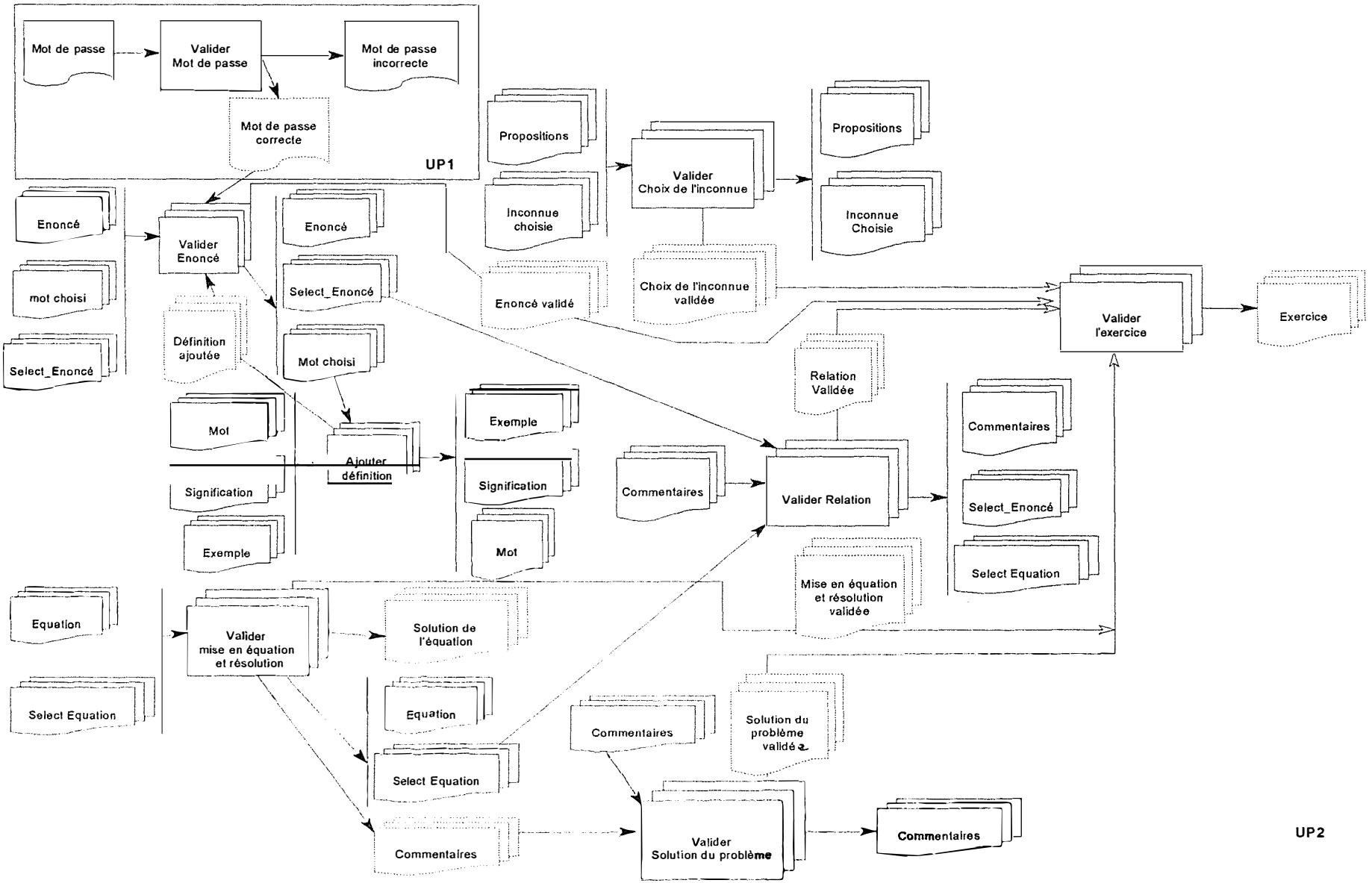
Si nous ne considérons que l'unité de présentation UP2, la saisie de données entre chaque fonction est asynchrone, c'est-à-dire que l'ordre d'exécution des actions est non prédéteminé, non-séquentiel. Il est toutefois synchrone lors de la validation de l'exercice. En effet, les fonctions *Valider Enoncé, Valider relation, Valider Solution du problème, Valider Choix de l'inconnue* et *Valider Mise en équation et résolution* doivent être exécutées pour que *Valider l'exercice* se réalise.

C. Le mode de déclenchement des fonctions

Globalement, le mode de déclenchement des fonctions est manuel explicite affiché car les fonctions doivent être déclenchées sur l'initiative de l'utilisateur à l'aide des actions prévues à cet effet. Ces actions se matérialiseront par des boutons de commande (par exemple, un bouton « OK »).

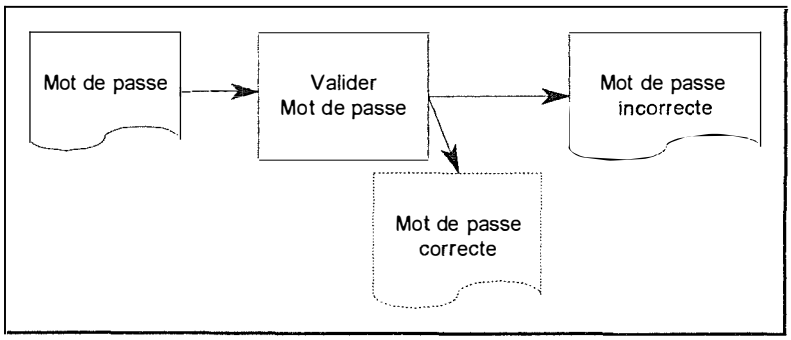
D. La métaphore

Nous avons essayé que l'interface ait une métaphore basée sur le mini-monde. Nous voulons que l'utilisateur agisse en tant qu'acteur et non en tant que donneur d'ordres à un robot.

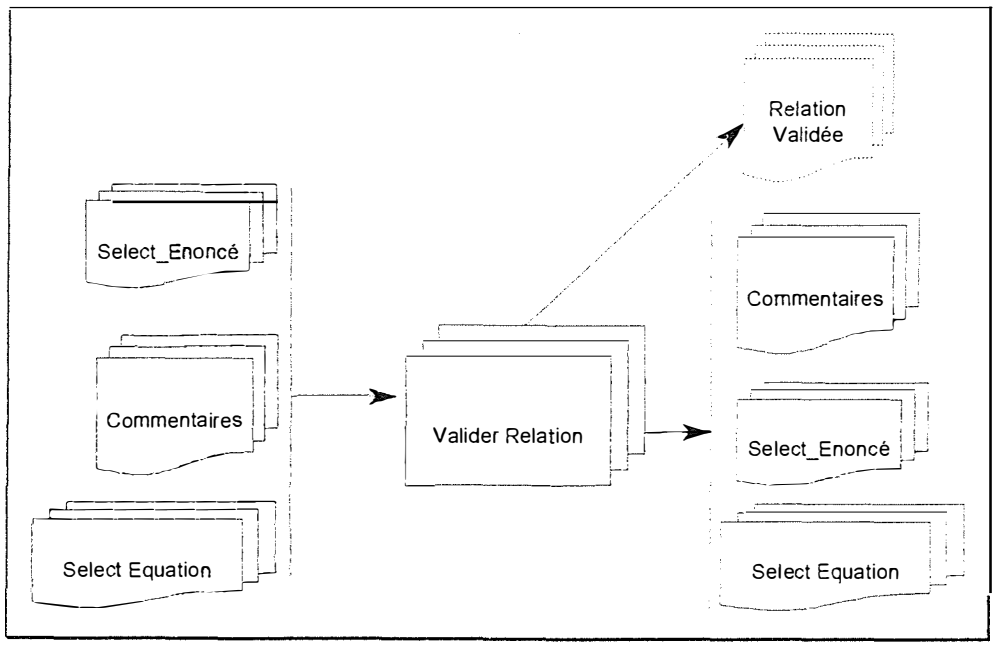

Mot de passe
Valider Mot de passe
Mot de passe incorrecte
Mot de passe correcte
UP1
Propositions
Valider Choix de l'inconnue
Propositions
Inconnue choisie
Inconnue Choisie
Enoncé
Valider Enoncé
Enoncé
Select_Enoncé
Enoncé validé
Choix de l'inconnue validée
Valider l'exercice
Exercice
mot choisi
Select_Enoncé
Définition ajoutée
Mot choisi
Relation Valldée
Mot
Ajouter définition
Exemple
Signification
Exemple
Signification
Mot
Commentaires
Valider Relation
Commentaires
Select_Enoncé
Select Equation
Mise en équation et résolution validée
Equation
Valider mise en équation et résolution
Solution de l'équation
Select Equation
Equation
Select Equation
Commentaires
Commentaires
Solution du problème validé
Valider Solution du problème
Commentaires
UP2

4.2 L'identification des fenêtres

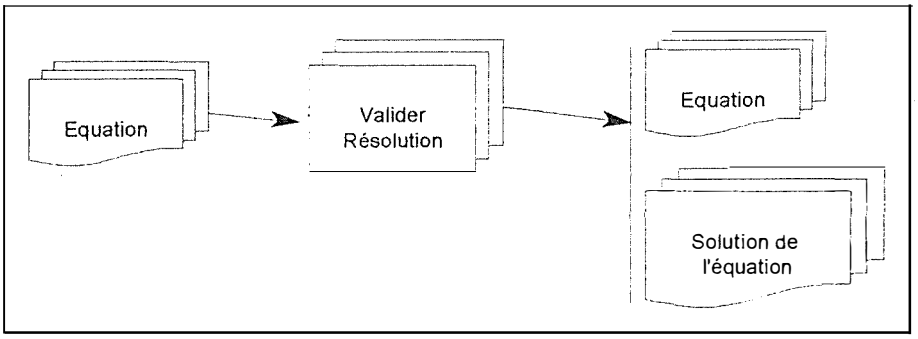
A partir du graphe d'enchaînement des fonctions sémantiques, décomposé en unités de présentation, nous avons fait correspondre une partition du sous-graphe de chaque UP en « sous-sous-graphe ». Chacun de ces « sous-sous-graphes » se matérialisera par une fenêtre.



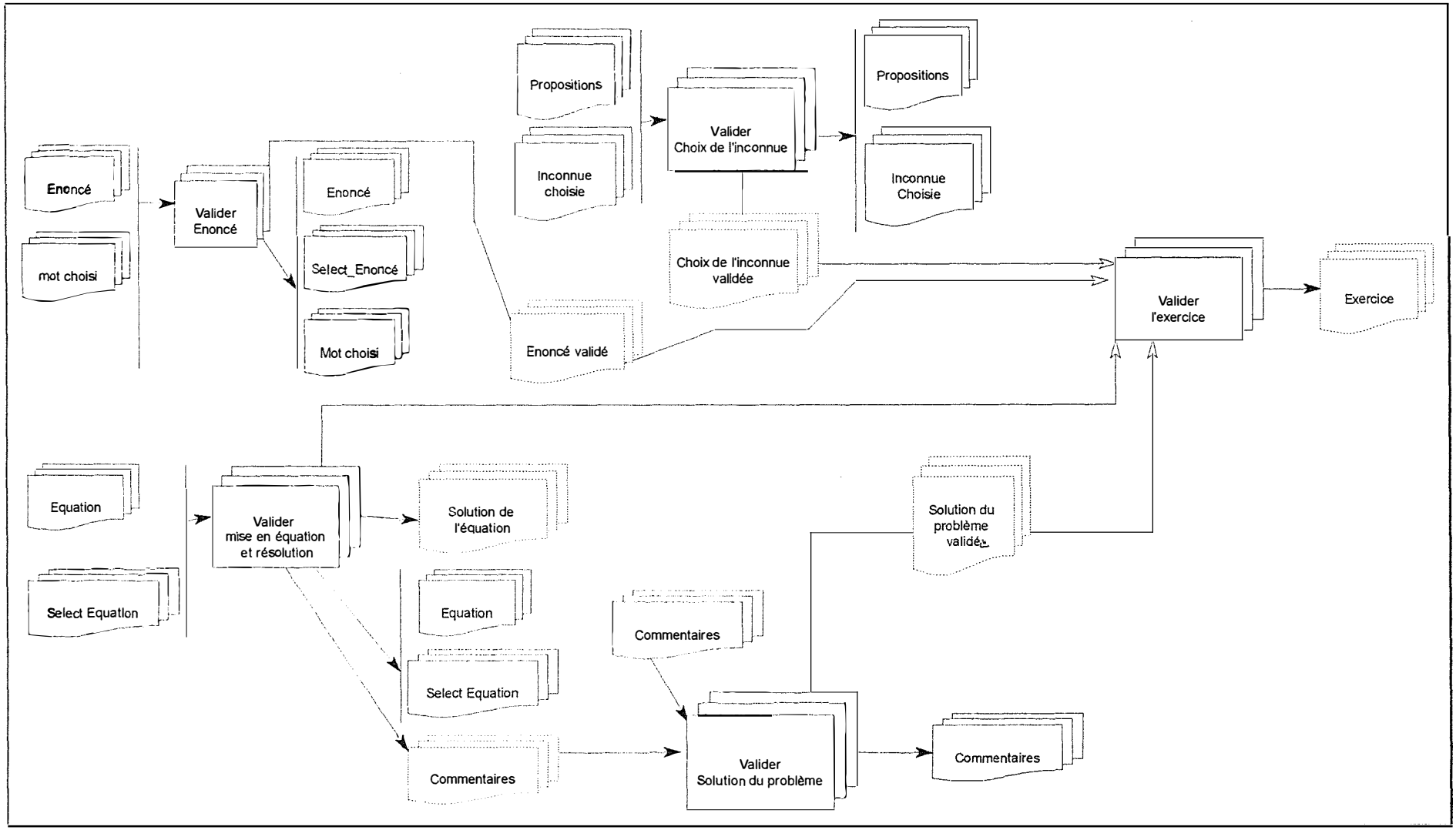
FEN 1 : Le mot de passe



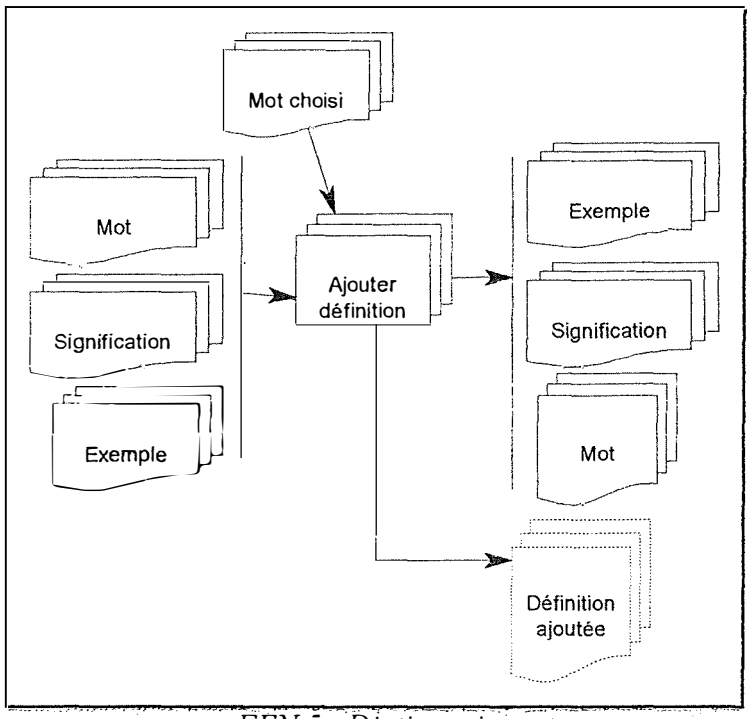
FEN 2 : Relation entre l'énoncé et l'équation



FEN 3 : Editeur d'équations propre au module de l'élève



FEN 4 : La fenêtre principale du module du professeur



FEN 5 : Dictionnaire

4.3 La sélection des objets interactifs abstraits (OIA)

4.3.1 La sélection des OIA de saisie

Au sein de la fenêtre « Le mot de passe », nous avons utilisé quatre barres de défilement. En effet, nous avons appliqué la règle ergonomique suivante « *Pour saisir une donnée numérique dans un intervalle continu connu apparenté à un réglage, à un contrôle, utiliser un curseur de défilement* » *[VAND92]*.

Dans notre fenêtre principale du module du professeur, nous avons suivi la règle ergonomique suivante, « *Pour saisir un groupe de données de types divers, utiliser une boîte de regroupement entourant les oia sélectionnés pour chacune des données du groupe* » *[VAND92]*. C'est ainsi que nous aurons, *Valider Enoncé, Valider Mise en équation et résolution, Valider Choix de l' inconnue, Valider Solution du problème* représentée par des boîtes de regroupement.

L'énoncé du problème, les propositions pour l'inconnue, les différents commentaires (solution du problème, relation entre énoncé et équation), la signification d'un mot et son exemple et l'éditeur d'équations seront saisis dans des champs d'édition multilinéaire car ils permettent à l'utilisateur d'introduire et de manipuler des chaînes de caractères en utilisant le clavier. Contrairement au champ d'édition uni-linéaire, la zone de saisie peut s'étendre sur plusieurs lignes et peut autoriser un défilement horizontal et/ou vertical.

L'équation, le mot à définir, la sélection dans l'équation, seront saisis dans des champs d'édition uni-linéaire.

Finalement, nous utiliserons une liste de combinaison déroulante pour la sélection dans l'énoncé car elle permet d'éviter la saturation de la présentation à l'écran dans le cas où l'utilisateur doit faire de multiples sélections discontinues en rapport avec une même sélection dans l'équation.

Remarque : Dans le module de l'élève, l'énoncé du problème, les propositions pour l'inconnue, les différents commentaires (solution du problème, relation entre énoncé et équation), l'éditeur d'équations, la signification d'un mot et son exemple sont des OIA d'affichage.

4.4.2 La sélection d'OIA d'affichage

Les fenêtres que nous venons de définir dans le point 4.2 constituent des fenêtres logiques. Nous pouvons dire que les fenêtres FEN 2 (Relation entre l'énoncé et l'équation), FEN 3 (L'éditeur d'équation) et FEN 5 (Dictionnaire), comportant des informations en entrée et/ou en sortie se matérialiseront par une boîte de dialogue. Ces boîtes de dialogue seront utilisées pour fournir de l'information à l'utilisateur et pour lui permettre d'introduire des données.

A l'intérieur des différentes fenêtres, nous retrouvons toujours,

- un libellé qui est le titre de la fenêtre
- trois boutons de commande servant à confirmer, annuler ou demander de l'aide, issus de la règle ergonomique suivante « *Toute boîte de dialogue doit posséder un bouton de validation, un bouton d'annulation et, si possible, un bouton d'aide* » [VAND92].

La solution de l'équation sera affichée dans un libellé.

Nous retrouverons également un ensemble d'objets de feed-back matérialisés par des messages d'information, utilisés pour fournir de l'information à l'utilisateur.

A l'intérieur de ces objets de feed-back, nous retrouvons,

- un libellé qui est le titre de la fenêtre du message;
- une icône unique et identifiante pour les différents types de message;
- un libellé expliquant la situation courante
- un bouton de commande sur lequel l'utilisateur peut appuyer pour confirmer sa lecture du message.

Remarques :

- Nous utiliserons un composant 'DBNavigator' offert par Delphi pour naviguer dans les données de nos bases de données. Il s'agit d'un composant graphique représentant des boutons proches d'un magnétocassette (métaphore du mini-monde).
- Dans le module de l'élève, la solution de l'équation est un OIA de saisie.

4.4 La transformation des OIA en OIC

Pour chaque OIA que nous avons défini dans le point précédent, nous allons faire correspondre les OIC propres à l'environnement physique de Microsoft Windows. Ainsi, nous obtiendrons,

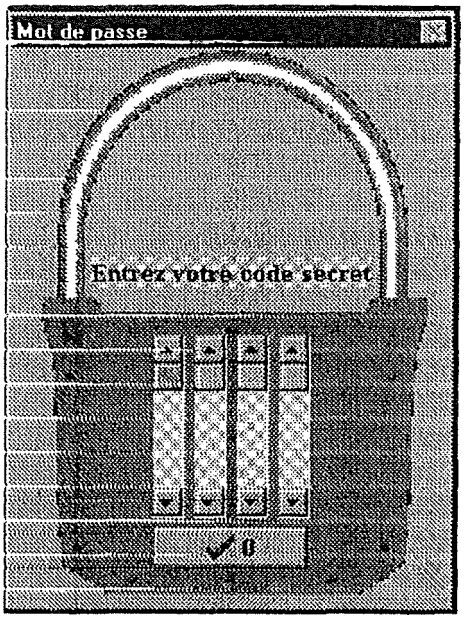
♦ Champ d'édition uni-linéaire	Edit box
♦ Bouton de commande	Push Button
♦ Libellé	Label
♦ Icône	Icon
♦ Liste de combinaison déroulante	Drop-down combination box
♦ Boîte de dialogue	Dialog box
♦ Message	Message
♦ Boîte de regroupement	Group box
♦ Barre de défilement	Scroll bar
♦ Champ d'édition multilinéaire	Multiple edit box
♦ Fenêtre	Window
♦ Fenêtre d'aide	Help window

4.5 Placement des Objets interactifs concrets au sein des fenêtres

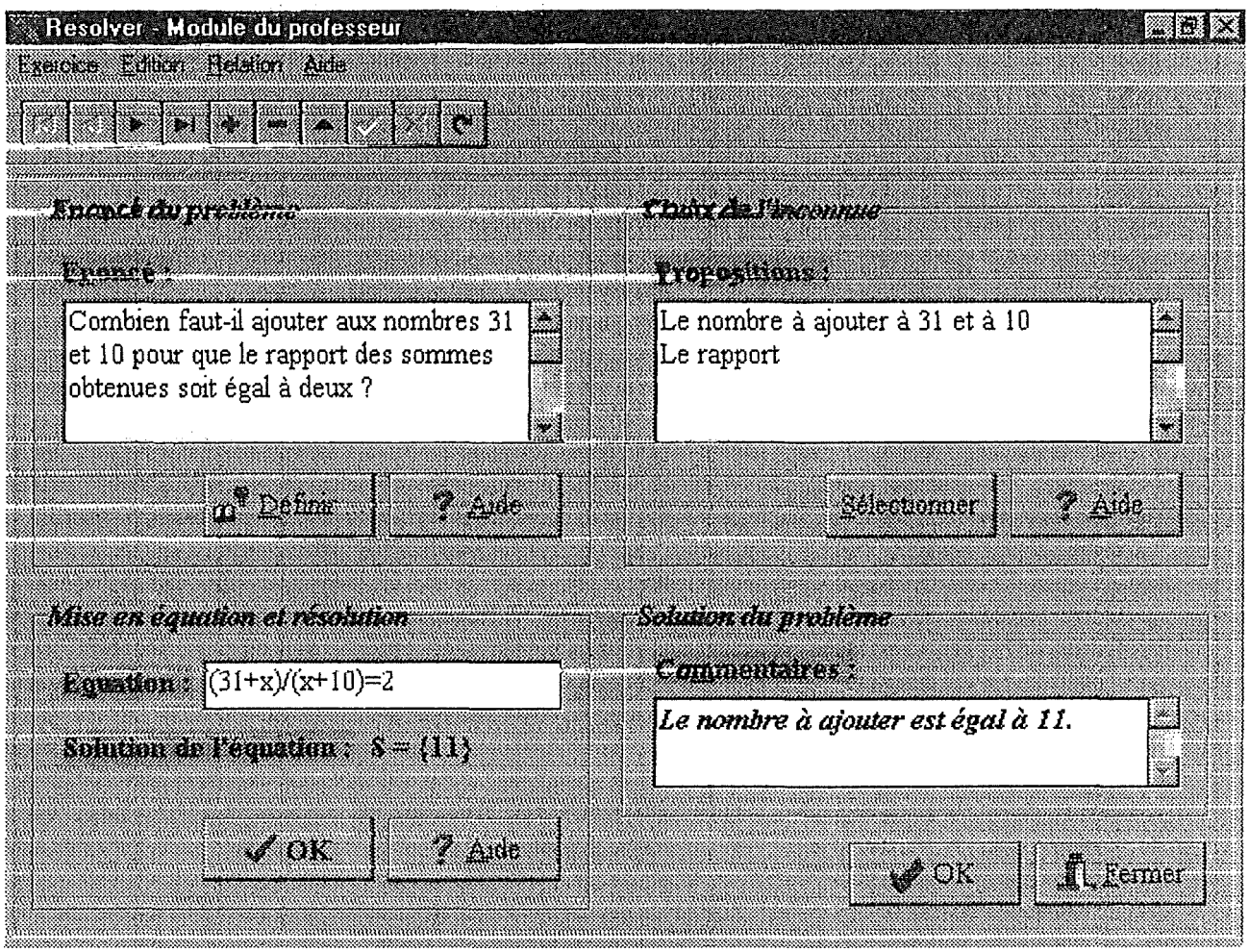
Nous allons décrire l'aspect des fenêtres que nous avons créées pour notre didacticiel. Nous avons essayé de respecter les règles de placement des objets inteactofs concrets au sein de chaque fenêtre. Les principales règles que nous citons ci-dessous sont toutes issues de [VAN92].

- *Les mnémoniques doivent être uniques et, s'ils sont disponibles, doivent être affichés, p 14.*
- *Toute boîte de dialogue doit posséder un bouton de validation, un bouton d'annulation et, si possible, un bouton d'aide, p44.*
- *Un libellé identificatif doit être prévu pour tout oic associé à une donnée à saisir, p 48.*
- *Tout libellé identificatif d'un oic de contrôle devrait posséder un mnémonique, p49.*
- *Le libellé, qui est l'entête d'une boîte de regroupement, doit être justifié à gauche avec les oic compris dans la boîte de regroupement situé le plus à gauche, p 57.*
- *Tout groupe de boutons de commande relatifs à un même ensemble logique de données doit être disposé soit en ligne au bas de l'oic dans lequel ils sont compris soit en colonne situé à droite de l'oic, si la solution précédente ne convient pas, p58.*
- *L'ordre de positionnement des boutons de commande doit être le plus proche possible de l'ordre suivant : (OK), (Annuler), (Aide), p58.*
- *Les boutons de commande disposés horizontalement doivent être équilibrés verticalement, p65.*

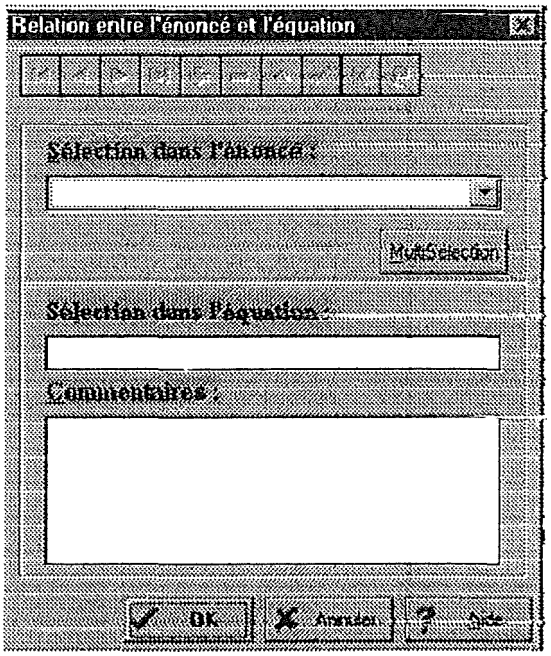
4.5.1 FEN 1 : Le mot de passe de l'unité de présentation UP1 pour le module du professeur.



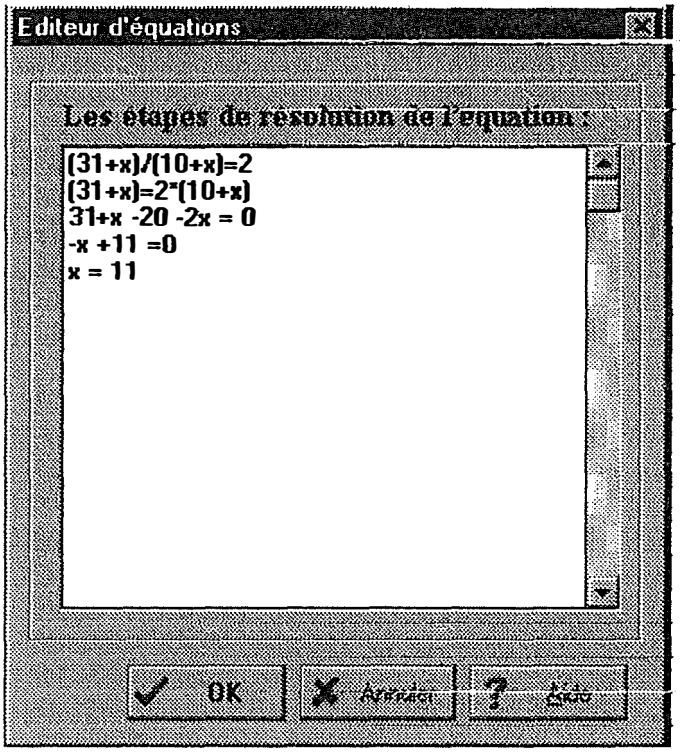
4.5.2 FEN 4 : La fenêtre principale du module du professeur (UP2)



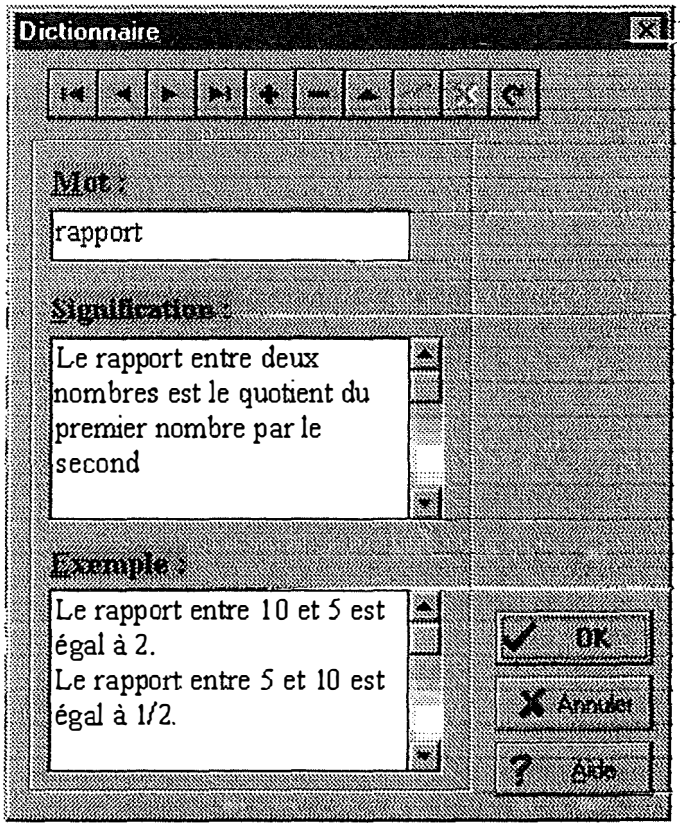
4.5.3 FEN 2 : La boîte de dialogue « Relation entre l'énoncé et l'équation » (UP2)



4.5.4 FEN 3 : La boîte de dialogue « Editeur d'équations » propre au module de l'élève

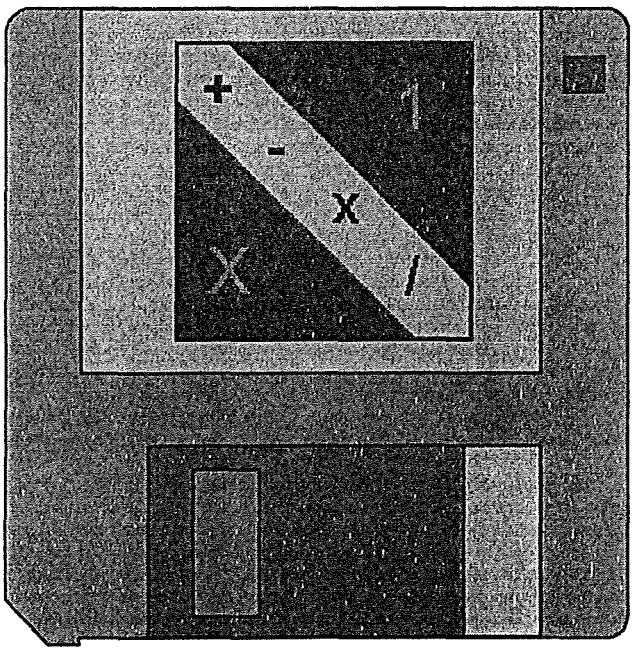


4.5.5 FEN 5 : La boîte de dialogue « Dictionnaire » (UP2)



Chapitre 5

L'analyse logicielle

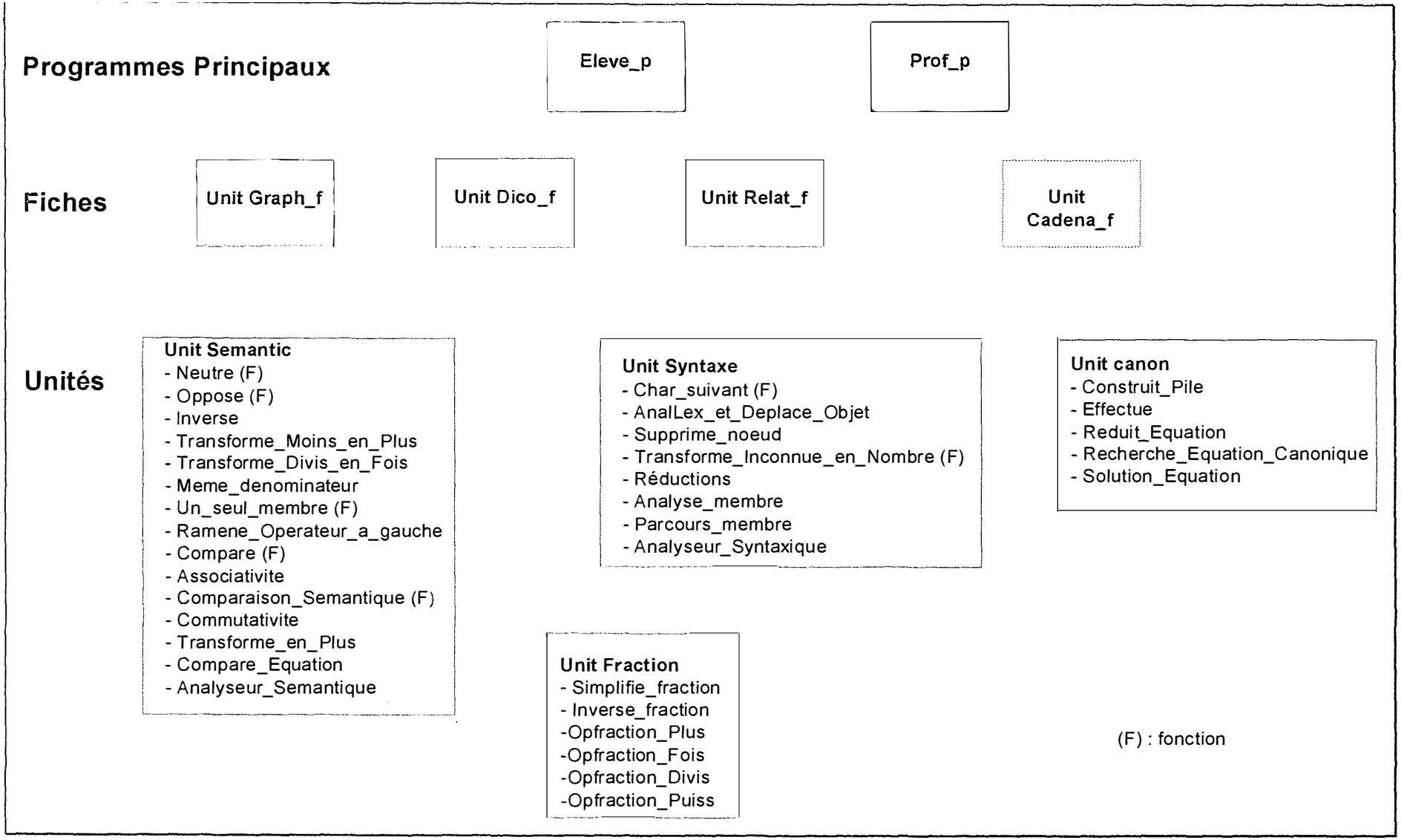


Introduction

A travers ce dernier chapitre, nous allons parcourir les diverses procédures et/ou fonctions qui demandent des explications supplémentaires aux commentaires fournis dans le listing. Ainsi, toute personne, qui souhaiterait comprendre, employer ou même modifier certaines procédures et/ou fonctions, pourrait se servir de ce chapitre comme un petit guide de l'implémentation de notre didacticiel.

Néanmoins, pour bien aborder ce chapitre, nous insistons sur la nécessité d'une bonne compréhension des concepts et des stratégies développés au chapitre 3. Des références vers ce chapitre seront réalisées tout au long de ce chapitre.

5.1 Les programmes principaux, les fiches et les unités



Le schéma de la page précédente reprend les différents programmes, fiches et unités que nous avons réalisés pour le fonctionnement du didacticiel. Le didacticiel est composé de deux modules, le module du professeur et le module de l'élève (cfr. 2.2.3. Une réflexion à partir d'un exemple, page 29). Pour chacun de ces modules, nous avons donc créé une application Delphi définie par un fichier projet (prof_p et eleve_p). Ce fichier est principalement constitué d'unités, de fiches et de fichiers d'aide. Il se charge de créer les fiches nécessaires (graph_f, dico_f, relat_f; cadena_f, encadrement pointillé, est utilisé uniquement par le fichier projet prof_p). Ensuite, il lance la fiche principale.

Le programme prof_p utilise une fiche supplémentaire par rapport au programme eleve_p. Il s'agit de la fiche cadena_f qui reprend en fait le mécanisme de vérification d'accès à un programme. Nous pourrions également utiliser un petit programme qui permettrait de crypter les données fournies par le professeur.

Nous vous renvoyons au chapitre 4 pour les détails concernant l'interface de chaque fiche. Dans la suite de ce travail, nous allons surtout approfondir quelques procédures et/ou fonctions relatives principalement à l'unité « Syntaxe ».

Remarques : Le dessin d'une fiche n'est pas une source Pascal. Il est contenu dans un fichier pouvant s'assimiler à un fichier de ressources (*.dfm). Le code source d'une fiche ou d'une unité sera sauvé dans un fichier *.pas tandis que les fichiers projet seront sauvés avec une extension *.dpr.

5.2 L'unité « Syntaxe »

5.2.1 La définition de notre grammaire

Avant d'entamer l'implémentation de l'analyseur syntaxique, la première étape est de définir notre grammaire. La connaissance de la syntaxe d'une expression mathématique est nécessaire pour l'utilisateur. Nous allons représenter, sous la forme B.N.F., l'ensemble des expressions qui seront admises et reconnues par notre didacticiel.

Les règles syntaxiques sont spécifiées dans le méta-langage suivant,

< S >	: 'entité quelconque de catégorie syntaxique s'
MG ::= MD	: 'MG est défini par MD'
a b	: 'a suivi de b'
a | b	: 'soit a soit b'

- < EQUATION > ::= < MEMBRE > = < MEMBRE>
- < MEMBRE > := < TERME > | < MEMBRE > < ADDOP > < TERME >
- < ADDOP > :: = + | -
- < TERME > ::= < FACTEUR > | < NBRESIGNE > < INCSIGNE >
  | < NBRESIGNE > < MULTOP > < INCSIGNE >
  | < TERME > < MULTOP > < NBRESIGNE >
  | < NBRESIGNE > < MULTOP > < NBRESIGNE >
  < INCSIGNE >
- < MULTOP > :: = * | /
- < FACTEUR > :: = < VALEUR > | < INCSIGNE > ^ 0 | < INCSIGNE > ^ 1
  | < INCSIGNE > ^ -1 | < NBRESIGNE > ^ < INT >
- < VALEUR > :: = < NBRESIGNE > | < INCSIGNE > | (< MEMBRE>)
  | (< MEMBRE>) | - (< MEMBRE>) | [< MEMBRE>] |
  | - [< MEMBRE>] | { < MEMBRE> } | - { < MEMBRE> }
- < NBRESIGNE > ::= < NBRE > | - < NBRE >
- < NBRE > ::= < INT > | < INT > < PTDEC > < CHIFFRES >
- < INCSIGNE > ::= < INC > | - < INC >
- < INC > ::= < LETTRE > | < LETTRE > < INC >

Lors de l'analyse lexicographique, nous construisons des « token », c'est-à-dire des éléments qui vont être transmis à l'analyseur syntaxique. La construction de ces « tokens » est très utile pour les retours en arrière. En effet, un « token » est une catégorie syntaxique atomique de notre grammaire. Lors d'un retour arrière, nous ne nous arrêtons pas à un caractère mais au « token » (chaîne de caractères).

- < INT > ::= 0 | < NCHIFFRE > | < NCHIFFRE > < CHIFFRES >
- < CHIFFRES > ::= < CHIFFRE > | < CHIFFRES > < CHIFFRE >
- < CHIFFRE > ::= 0 | < NCHIFFRE >
- < NCHIFFRE> ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
- < LETTRE > :: = a | b | c | d | e | f | g | h | i | j | ... | X | Y | Z
- < PTDEC > ::= . | ,

L'avantage d'écrire notre grammaire sous la forme B.N.F. réside dans le fait qu'il est possible d'ajouter de nouvelles règles syntaxiques plus facilement. L'idéal serait que le professeur puisse, lui-même, modifier, paramétrer la grammaire par rapport à ses propres exigences. Par exemple, accepte-t-il qu'un élève mette un point entre les décimales ou bien préfère-t-il une virgule.

L'application d'une règle syntaxique permet de remplacer une des séquences des éléments du membre de droite par le membre de gauche (ou inversement).

Remarques :

♦ < TERME > ::= < FACTEUR > | < NBRESIGNE > < INCSIGNE >
| < NBRESIGNE > < MULTOP > < INCSIGNE >
| < TERME > < MULTOP > < NBRESIGNE >
| < NBRESIGNE > < MULTOP > < NBRESIGNE >
< INCSIGNE >

Nous permettons que l'utilisateur omette l'opérateur « * » entre un <NBRESIGN> et < INCSIGNE>. Nous n'autorisons pas qu'un terme, contenant déjà l'inconnue, soit multiplié ou divisé par l'inconnue.

♦ < FACTEUR > :: = < VALEUR > | < INCSIGNE > ^ 0 | < INCSIGNE > ^ 1
| < INCSIGNE > ^ -1 | < NBRESIGNE > ^ < INT >

Etant donné que notre didacticiel ne se préoccupe que de problèmes du premier degré à une inconnue, notre grammaire n'autorisera uniquement qu'un nombre soit élevé à une puissance entière (positive ou négative), qu'une inconnue soit élevée à la puissance 0 (cela revient à dire qu'il s'agit du nombre 1), à la puissance 1 (premier degré) ou à la puissance -1 (si nous divisons un nombre par une inconnue).

5.2.2 La conception de l'analyseur syntaxique

L'analyseur lexicographique va transmettre à l'analyse syntaxique des symboles terminaux appartenant à l'ensemble V_T qui est constitué de + - * / ^ . , $[23] = ( ) { } [ ] 0 1 2 3 4 5 6 7 8 9 a b c d e ... X Y Z. Nous n'allons pas attendre que l'analyse lexicographique soit terminée avant de créer l'arbre de décomposition syntaxique. En d'autres termes, nous n'allons pas prendre en considération l'expression en entier.

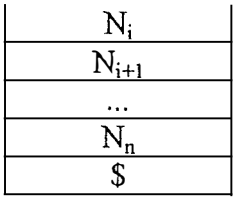
Nous lisons, de gauche à droite, la chaîne ω (N_1 N_2 ... N_{i-1} N_i ... N_n $) à analyser syntaxiquement, c'est-à-dire l'expression introduite par l'utilisateur. Ensuite, nous effectuons des réductions. Nous pourrions gérer la chaîne réduite à l'aide d'une seule pile. Les opérations de réduction consistent à supprimer des éléments dans celle-ci. Les tests sur ce qui précède ou sur ce qui suit, se font très simplement en consultant les premiers symboles de la pile.

[23] Le symbole « $ » est ajouté à la fin de l'expression introduite par l'utilisateur. Dès que l'analyseur lexicographique transmet ce symbole à l'analyseur syntaxique, ce dernier sait qu'il s'agit du dernier lexème.

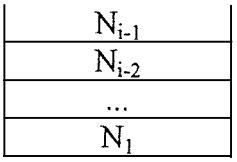
Nous avons choisi de construire l'arbre de décomposition syntaxique d'une autre manière. Nous souhaitons distinguer l'analyse lexicographique (construction d'un « token ») de l'analyse syntaxique. Pour ce faire, nous utilisons une autre pile qui contient cette fois des arbres (c'est-à-dire des pointeurs sur des noeuds). Les deux piles sont traitées en parallèle, la réduction par une règle a pour effet, sur la deuxième, de construire un nouvel arbre dont les fils se trouvent en tête de la pile, puis de remettre le résultat dans celle-ci.

5.2.2.1 La structure des deux piles

La première pile **ARBRE_LEXICO** contient tous les noeuds, formés de symboles terminaux, qui n'ont pas encore été pris en compte pour l'application d'une règle syntaxique.



La seconde pile **ARBRE_SYNTAXIQUE** sert à unir les noeuds formant la chaîne ω. Nous allons regarder dans notre grammaire s'il existe une règle qui soit applicable au premier élément (N_{i-1}) de la pile.



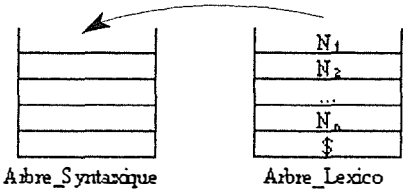
5.2.2.2 L'algorithme de construction de l'arbre de décomposition syntaxique

Pour construire l'arbre de décomposition syntaxique, nous allons effectuer plusieurs opérations sur les deux piles.

A. Première opération : Déplacer un élément

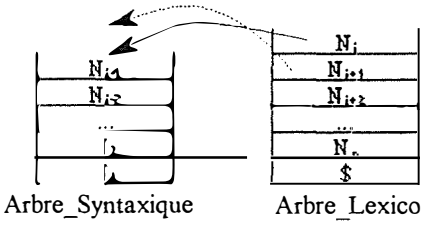
Une de ces opérations consiste à extraire l'élément se trouvant au sommet de la pile ARBRE_LEXICO et à le placer au sommet de la pile ARBRE_SYNTAXIQUE (cfr. la procédure *AnalLex_et_Deplace_Objet*, page 89). Quand se réalise-t-elle ?

- Lorsque la pile ARBRE_SYNTAXIQUE est vide;



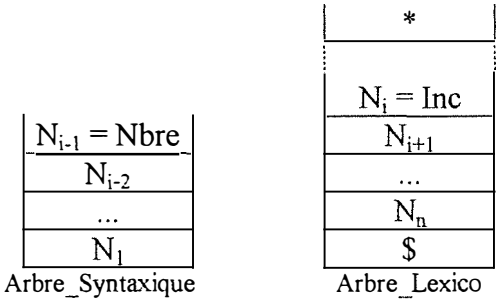
♦ Lorsque les éléments se trouvant au sommet des deux piles, ARBRE_LEXICO et ARBRE_SYNTAXIQUE, se présentent, dans le même ordre, dans le membre de droite d'une règle syntaxique.

< Categorie1 > :: = N_{i-1} N_i N_{i+1}



B. Deuxième opération : Ajouter un élément

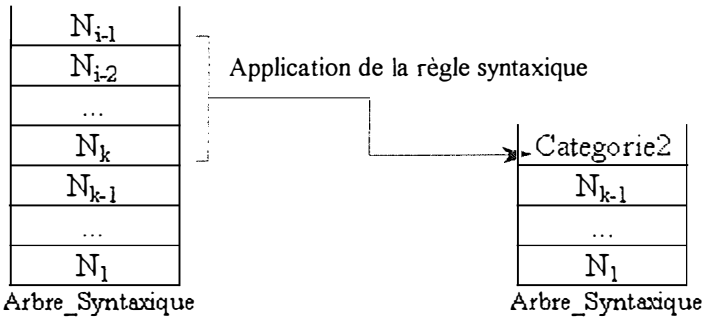
Lorsque l'utilisateur a omis de mettre un opérateur entre un nombre et l'inconnue, en introduisant son expression mathématique, nous considérons qu'il s'agit de l'opérateur « * ». Donc, si au sommet des piles ARBRE_LEXICO et ARBRE_SYNTAXIQUE, nous avons un élément représentant respectivement la catégorie syntaxique < INCSIGNE> et la catégorie syntaxique <NBRESIGN>, nous ajoutons un nouvel élément, le symbole terminal « * », au sommet de la pile ARBRE_LEXICO (cfr. la procédure *Ajoute_Fois* dans le listing).



C. Troisième opération : Réduire

Lorsque nous remplaçons un membre de droite d'une règle syntaxique par son membre de gauche, nous réalisons une *réduction*. D'un point de vue algorithmique, cela signifie que nous modifions la pile et réduisons sa hauteur (cfr. la procédure *Reductions*, page 92).

<Categorie2> :: = N_k ... N_{i-2} N_{i-1}



Nous voyons que les derniers éléments N_k ... N_{i-2} N_{i-1}, placés dans la pile ARBRE_SYNTAXIQUE (c'est-à-dire ceux se situant au sommet), correspondent à la partie droite de la règle syntaxique. Nous les remplaçons, dans la pile ARBRE_SYNTAXIQUE, par l'élément *Catégorie2* (partie gauche de cette même règle).

Nous pouvons donc effectuer une réduction si nous trouvons une suite d'éléments, situés en début de pile, qui soient identiques à ceux composant la partie droite d'une des règles syntaxiques. Cependant, une seconde condition doit être remplie pour réaliser une réduction. Il ne faut pas qu'il soit possible de réaliser la première opération « *Déplacer un élément* ». Cela signifie que les éléments se trouvant au sommet des deux piles, ARBRE_LEXICO et ARBRE_SYNTAXIQUE, ne doivent pas former une partie d'un membre de droite d'une règle syntaxique

INVARIANT DE L'ALGORITHME DE CONSTRUCTION DE L'ARBRE DE DECOMPOSITION SYNTAXIQUE

Si l'expression introduite par l'utilisateur est syntaxiquement correcte, alors, à chaque étape de construction de l'arbre de décomposition syntaxique, les différents éléments de la pile ARBRE_SYNTAXIQUE, du plus ancien (N_1) au plus récent (N_{i-1}), ainsi que ceux de la pile ARBRE_LEXICO N_i, ...,N_n, forment une *proposition* de cette expression (cfr. Chapitre 3 Les concepts et les stratégies mis en jeu, page 46).

Cependant, il se peut que l'expression soit incorrecte au niveau sémantique. Par exemple, un des caractères de l'expression ne fait pas partie du vocabulaire terminal V_T de notre grammaire; ou bien, nous ne parvenons pas à réduire la pile ARBRE_SYNTAXIQUE, ... Nous devons être capables de localiser l'erreur et d'envoyer un message approprié à l'utilisateur.

5.2.2.3 La table de décision

A la page suivante, nous avons défini une table dont l'utilité est de mettre en évidence le type de décision que l'analyseur syntaxique doit prendre lorsque le sommet de la pile ARBRE_SYNTAXIQUE contient tel élément et le sommet de la pile ARBRE_LEXICO contient tel autre élément.

Chaque couple (Arbre_Syntaxique, Arbre_Lexico) a une image unique dans la table. Les différentes valeurs que nous rencontrons dans la table sont :

- A : Opération *Ajouter un élément*
- D : Opération *Déplacer un élément*
- R : Opération *Réduire*
- E, P renvoie un message d'erreur approprié à l'utilisateur

Nous avons construit cette table de décision en nous basant, d'une part, sur les différentes conditions pour réaliser les trois opérations et d'autre part, sur notre grammaire.

Chaque ligne représente la valeur que peut prendre l'élément situé au sommet de la pile ARBRE_SYNTAXIQUE tandis que chaque colonne représente la valeur que peut prendre l'élément formant le sommet de la pile ARBRE_LEXICO. Nous avons moins de colonnes que de lignes puisque la pile ARBRE_LEXICO ne contient que des symboles terminaux.

Arbre_Lexico / Arbre_Syntaxique	< Inc >	< Nbre >	+	-	*	/	^	(	)	< Rien >
< Membre >	E	E	D	D	E	E	E	E	D	R
< Terme >	E	E	R	R	D	D	E	E	R	R
< Facteur >	E	E	R	R	R	R	D	E	R	R
< Valeur >	E	E	R	R	R	R	R	E	R	R
< Inc >	E	E	R	R	R	R	R	E	R	R
< Nbre >	A	E	R	R	R	R	R	E	R	R
+	D	D	E	D	E	E	E	D	E	E
-	D	D	E	D	E	E	E	D	E	E
*	D	D	E	D	E	E	E	D	E	E
/	D	D	E	D	E	E	E	D	E	E
^	P	D	E	D	E	E	E	D	E	E
(	D	D	E	D	E	E	E	D	E	E
)	E	E	R	R	R	R	R	E	R	R
< Rien >	D	D	E	D	E	E	E	D	E	E

5.2.3 L'implémentation de l'analyseur syntaxique

Procedure AnalLex_et_Deplace_Objet

(Var Symbole : char; Var Msg : Pchar; Var Erreur : boolean;
Var Arbre_Syntaxique : Parbre; Var Arbre_Lexico : Parbre;
Var Fin_Membre : boolean; Num_Mbre : integer);

Résultats

Symbole	: Char	% Caractère extrait du string *Equ_dollar*, compatible avec un symbole terminal de notre grammaire.
Msg	: PChar	% Message de guidage pour l'utilisateur
Erreur	: Boolean	% Indicateur utilisé pour signaler que la syntaxe du string *Equ_dollar* est incorrecte lorsqu'il est initialisé à *False*.
Arbre_Syntaxique	: PArbre	% Pile en cours de construction représentant un membre de l'équation. Elle contient les noeuds déjà analysés lexicographiquement et chaînés.
Arbre_Lexico	: PArbre	% Pile composée de noeuds, constituée essentiellement de symboles terminaux auxquels nous n'avons pas encore essayé d'appliquer une règle syntaxique.
Fin_Membre	: Boolean	% Indicateur de fin d'analyse d'un membre
Num_Membre	: Integer	% Indicateur permettant de connaître le membre de l'équation qui est en cours d'analyse.

Variables globales

Equ_dollar	: String	% Expression mathématique introduite par l'utilisateur à laquelle nous avons ajouté le caractère '$' pour en signaler la fin.
i	: Integer	% Indice du string *Equ_dollar*.

Variables intermédiaires

Bouge_Objet	: Boolean	% Indicateur de déplacement de l'élément de la pile *Arbre_Lexico*, vers la pile *Arbre_Syntaxique*.
Temp_Categorie	: TCategorie	% Symbole terminal dont est issu l'élément en cours d'analyse
Temp_Nbre	: TFraction	% Valeur numérique de l'élément en cours d'analyse lexicographique. Le symbole terminal est un nombre.
Temp_Coefficient	: TFraction	% Valeur numérique du coefficient de l'élément en cours d'analyse lexicographique. Le symbole terminal est une expression contenant l'inconnue.
Ajout_Noeud	: PArbre	% Variable dynamique temporaire utilisée lorsque nous ajoutons des éléments à la pile *Arbre_Syntaxique* (Arbre_Lexico^.suivant < > nil).
Temp_Degre	: integer;	% Valeur numérique de la puissance de l'élément en cours d'analyse lexicographique. Le symbole terminal est un terme contenant l'inconnue.

Constantes

Lettres : set of char = ['a' .. 'z', 'A' .. 'Z'] % Ensemble des lettres de l'alphabet (minuscules et majuscules)

Chiffres : set of char = ['0'..'9',',','.'] % Ensemble des caractères pouvant apparaître dans un nombre

Effet

Premièrement, elle analyse lexicographiquement le string *Equ_dollar* et en extrait un caractère *Symbole* compatible avec un symbole terminal de notre grammaire. Deuxièmement, elle ajoute un élément de la pile *Arbre_Lexico* à la pile *Arbre_syntaxique*. Ensuite, l'élément analysé lexicographiquement *Symbole*, avec ses attributs propres, est ajouté à la pile *Arbre_Lexico* (cfr. A. Première opération : Déplacer un élément, page 85).
Si l'équation est syntaxiquement incorrecte, la procédure renvoie un message d'erreur *Msg* à l'utilisateur et met à vrai la variable booléenne *Erreur*.

Préconditions

Symbole = Equ_dollar[i-1]
0 < *i* < = Length (Equ_dollar) +1
Num_Membre = 1 ou *Num_Membre* = 2
Arbre_Lexico et *Arbre_Syntaxique* sont non vides
Erreur est mis à faux

Postconditions

0 < *i'* < = i +1< = Length (Equ_dollar)
Symbole' = Equ_dollar [i'-1]
Arbre_syntaxique' et *Arbre_Lexico'* ont été modifiés; *Fin_Membre'* est modifié si Equ_dollar [i-1] était égale à '=' ou '$'

Exceptions :

(1) Si la procédure Reforme_Nombre renvoie un message d'erreur à l'utilisateur
(2) Si la procédure Repere_Inconnue renvoie un message d'erreur à l'utilisateur
(3) Si Symbole est '=' et Num_Mbre=2
(4) Si Symbole est '=', Num_Mbre=1 et (Arbre_lexico^.Categorie_Evol = Rien)
(5) Si Symbole est '=', Num_Mbre=1 et (Arbre_lexico^.Categorie_Evol dans [Plus .. ParG])
(6) Si Symbole est '$', Num_Mbre=1
(7) Si Symbole est '$', Num_Mbre=2 et (Arbre_Lexico^.Categorie_evol = Rien)
(8) Si Symbole n'est pas un symbole terminal ou '$', ' '

Msg contient un message de guidage pour l'utilisateur
Erreur est mis à vrai

Procédure utilisée par la procédure Analyse_Membre

Procédure utilisant

- Char_Suivant
- Reforme_Nombre
- Repere_Inconnue

Algorithme

Si (Arbre_Lexico^.Suivant = nil)
Alors Bouge_Objet ← faux
Tant que Bouge_Objet est faux
Si le symbole que nous sommes en train d'analyser lexicographiquement est

- '+', '-', '*', '/','^', '{', '[', '(', ')', ']', '}'
  Alors Bouge_Objet est mis à vrai
  Symbole contient le caractère suivant d'Equ_dollar
  Temp_Categorie a, selon la nature du symbole terminal, la valeur suivante :
  Plus ('+'), Moins ('-'), Fois ('*'), Divis ('/'), Puiss ('^'),
  ParG ('{', '[', '('), ParD ('}', ']', ')')
- '=' ,
  Alors **Si** nous sommes dans le second membre
  Alors cela signifie que l'utilisateur introduit pour la seconde fois le signe d'égalité. L'équation est donc syntaxiquement incorrecte.
  Sinon, Si (Arbre_lexico^.Categorie_Evol = Rien)
  Alors cela veut dire que l'utilisateur a commencé son équation par le symbole d'égalité, il a une erreur syntaxique
  Sinon Si (Arbre_lexico^.Categorie_Evol est un opérateur ou une parenthèse ouverte)
  Alors, il y a une erreur syntaxique
  Bouge_Objet et Fin_Membre sont mis à vrai
  Temp_Categorie ← Rien
- '$' ,
  Alors, Si nous sommes dans le premier membre
  Alors Si (arbre_lexico^.Categorie_evol = Rien)
  Alors l'utilisateur n'a pas entré d'équation
  Sinon l'équation est imcomplète
  Dans les deux cas, l'équation est syntaxiquement incorrecte
  Sinon Si (Arbre_Lexico^.Categorie_evol = Rien)
  Alors L'équation est incomplète. Elle n'a pas de second membre.
  Bouge_Objet et Fin_Membre sont mis à vrai
  Temp_Categorie ← Rien
- Une lettre,
  Alors, nous vérifions s'il s'agit de l'inconnue
  Bouge_Objet est mis à vrai
- Un chiffre
  Alors, nous reformons éventuellement le nombre
  Bouge_Objet est mis à vrai
- ' ' (espace)
  Alors, nous passons au caractère suivant
- Autre caractère
  Alors, il y a une erreur syntaxique car le caractère ne fait pas partie de la syntaxe définie par notre grammaire.

Si nous n'avons pas trouvé d'erreur syntaxique, alors nous déplaçons l'élément au sommet de la pile Arbre_Lexico vers le sommet de la pile Arbre_syntaxique. Ensuite, nous ajoutons au sommet de la pile Arbre_Lexico, les attributs du caractère (ou de la suite de caractères) que nous venons d'analyser.

Sinon {Arbre_Lexico^.Suivant ≠ nil}
Ajout_Noeud ← Arbre_Lexico^.Suivant;
Arbre_Lexico^.Suivant ← Arbre_Syntaxique;
Arbre_Syntaxique ← Arbre_Lexico;
Arbre_Lexico ← Ajout_Noeud

Procedure Reductions

(Var Arbre_Syntaxique : Parbre; Var Msg : Pchar; Var Erreur : boolean);

Résultats

Arbre_Syntaxique	: PArbre	% Pile en cours de construction représentant un membre de l'équation. Elle contient les noeuds déjà analysés lexicographiquement et chaînés.
Msg	: PChar	% Message de guidage pour l'utilisateur
Erreur	: Boolean	% Indicateur utilisé pour signaler que la syntaxe du string *Equ_dollar* est incorrecte lorsqu'il est initialisé à *False*.

Effet

Cette procédure effectue l'opération *Réduire* que nous avions décrite précédemment (cfr. C. Troisième opération : Réduire, page 86). Elle applique la règle syntaxique *Recherche_Regle* aux éléments formant le sommet de la pile *Arbre_Syntaxique.* Si aucune de ces règles ne convient, l'équation est syntaxiquement incorrecte. La procédure renvoie un message d'erreur *Msg* à l'utilisateur et met à vrai la variable booléenne *Erreur.*

Préconditions

Erreur est mis à faux
Arbre_Syntaxique est non vide

Postcondition

Arbre_Syntaxique' a été modifié.

Exceptions :

(1) Si la procédure Recherche_Regle renvoie un message d'erreur à l'utilisateur
(2) Si les procédures Reduct_5, Reduct_7 et Reduct_8 renvoie un message d'erreur à l'utilisateur
(3) Si nous ne trouvons pas une règle syntaxique applicable aux éléments formant le sommet de la pile Arbre_Syntaxique (Recherche_Regle = 0).

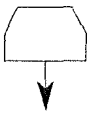
Msg contient un message de guidage pour l'utilisateur
Erreur est mis à vrai

Procédure utilisée par la procédure Analyse_Membre

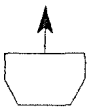
Procédure utilisant

- Recherche_Regle
- Reduct_I où I = 1, 3, 5, 7, 8, 10, 12, 13

Algorithme



Si le numéro *Recherche_Regle* est
0 : Il y a une erreur syntaxique dans l'équation
1 : Reduct_1; { <Valeur> ::= - <Inc> }
2 : Arbre_Syntaxique^.Categorie_Evol ← Valeur; { <Valeur> ::= <Inc> }
3 : Reduct_3; { <Valeur> ::= - <Nbre> }
4 : Arbre_Syntaxique^.Categorie_Evol ← Valeur; { <Valeur> ::= <Nbre> }
5 : Reduct_5; { <Facteur> ::= <Facteur> ^ <Valeur> }
6 : Arbre_Syntaxique^.Categorie_Evol ← Facteur; { <Facteur> ::= <Valeur> }
7 : Reduct_7; { <Terme> ::= <Terme> * <Facteur> }
8 : Reduct_8; { <Terme> ::= <Terme> / <Facteur> }
9 : Arbre_Syntaxique^.Categorie_Evol ← Terme; { <Terme> ::= <Facteur> }
10 : Reduct_10; { <Membre> ::= <Membre> <addop> <Terme> }
11 : Arbre_Syntaxique^.Categorie_Evol ← Membre; {<Membre>::= <Terme>}
12 : Reduct_12; { <Valeur> ::= - (<Membre>) }
13 : Reduct_13; { <Valeur> ::= (<Membre>) }



A présent, nous allons détailler la fonction *Recherche_Regle* et les procédures de réduction *Reduct_5*, *Reduct_8*.

Function Recherche_Regle : integer;

Variables globales

Arbre_Syntaxique	: PArbre	% Pile en cours de construction représentant un membre de l'équation. Elle contient les noeuds déjà analysés lexicographiquement et chaînés.
Msg	: PChar	% Message de guidage pour l'utilisateur
Erreur	: Boolean	% Indicateur utilisé pour signaler que la syntaxe du string *Equ_dollar* est incorrecte lorsqu'il est initialisé à *False*.

Variables intermédiaires

Categorie_Prec	: TCategorie	% Catégorie syntaxique
Numero	: Integer;	% Numéro identifiant la règle syntaxique que nous pouvons appliquer aux éléments formant le sommet de la pile *Arbre_Syntaxique*.

Effet

Cette fonction recherche le numéro de la règle syntaxique qu'il faudra appliquer aux éléments formant le sommet de la pile *Arbre_Syntaxique*.

Précondition

Arbre_Syntaxique est non vide

Postcondition

Recherche_Regle' a une valeur entre 0 et 13.

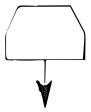
Exception :

Si l'utilisateur a oublié d'introduire une parenthèse

Msg contient un message de guidage pour l'utilisateur
Erreur est mis à vrai

Procédure utilisée par la procédure Reductions

Algorithme

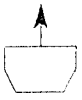


numero ← 0
Si Arbre_Syntaxique^.Categorie_Evol est

- une expression contenant l'inconnue ou un nombre,
  Alors, Si Arbre_Syntaxique^.suivant^.Categorie_Evol est l'opérateur '-'
  Alors Categorie_Prec ← Arbre_Syntaxique^.suivant^.suivant^.Categorie_evol
  Si Categorie_Prec est un opérateur, une parenthèse ouverte ou rien
  Alors numero ← 1(expression contenant l'inconnue) ou 3 (nombre)
  Si (numero = 0) **Alors** numero ← 2 (expression contenant l'inconnue) ou 4 (nombre)
- une valeur
  Alors, Si Arbre_Syntaxique^.suivant^.suivant^.Categorie_Evol = Facteur
  et Arbre_Syntaxique^.suivant^.Categorie_Evol est l'opérateur '^'
  Alors numero ← 5
  Sinon numero ← 6
- un facteur
  Alors, Si Arbre_Syntaxique^.suivant^.suivant^.Categorie_Evol = Terme
  Alors Si Arbre_Syntaxique^.suivant^.Categorie_evol est
  - l'opérateur '*', numero ← 7
  - l'opérateur '/', numero ← 8

  Si (numero = 0) **Alors** numero ← 9
- un terme
  Alors, Si Arbre_Syntaxique^.suivant^.suivant^.Categorie_Evol = Membre
  et Arbre_Syntaxique^.suivant^.Categorie_Evol est l'opérateur '+' ou '-'
  Alors numero ← 10
  Si (numero = 0) **Alors** numero ← 11
- une parenthèse fermée
  Alors, Si Arbre_Syntaxique^.suivant^.suivant^.Categorie_Evol est une parenthèse ouverte
  et Arbre_Syntaxique^.suivant^.Categorie_Evol = Membre
  Alors Si Arbre_Syntaxique^.suivant^.suivant^.suivant^.Categorie_Evol est l'opérateur '-'
  Alors Categorie_Prec
  ← Arbre_Syntaxique^.suivant^.suivant^.suivant^.suivant^.Categorie_Evol
  Si Categorie_Prec est un opérateur, une parenthèse ouverte ou rien
  Alors numero ← 12
  Si (numero = 0) **Alors** numero ← 13
  Sinon les parenthèses sont mal placées. Il y a une erreur syntaxique

Recherche_Regle:=numero



Procedure Reduct_5 ***{ <Facteur> ::= <Facteur> ^ <Valeur> }***

Variables globales

Msg	: PChar	% Message de guidage pour l'utilisateur
Erreur	: Boolean	% Indicateur utilisé pour signaler que la syntaxe du string *Equ_dollar* est incorrecte lorsqu'il est initialisé à *False.*
Arbre_Syntaxique	: PArbre	% Pile en cours de construction représentant un membre de l'équation. Elle contient les noeuds déjà analysés lexicographiquement et chaînés.

Variables intermédiaires

Ptr1	: PArbre	% Pointeur vers l'élément N_n placé au sommet de la pile *Arbre_Syntaxique.*
Ptr2	: PArbre	% Pointeur vers l'avant-dernier élément (N_{n-1}) placé au sommet de la pile *Arbre_Syntaxique.*
Ptr3	: Parbre;	% Pointeur vers l'élément (N_{n-2}) de la pile *Arbre_Syntaxique.*

Effet

Cette procédure crée un nouvel arbre de type <Facteur> (catégorie syntaxique) dont les attributs seront déterminés en fonction de la nature des deux éléments <Valeur> et <Facteur>.

Préconditions

Arbre_Syntaxique, *Arbre_Syntaxique*^.suivant, *Arbre_Syntaxique*^.suivant^.suivant sont non vides.
Erreur est mis à faux

Postcondition

Arbre_Syntaxique' a été modifié, *Arbre_Syntaxique*^.*Categorie_Evol'* = Facteur

Exceptions :

(1) Si la procédure OpFraction_Puiss renvoie un message d'erreur à l'utilisateur
(2) Si l'expression contenant l'inconnue est élevée à une puissance différente de -1, 0, et de 1.

Msg contient un message de guidage pour l'utilisateur
Erreur est mis à vrai

Procédure utilisée par la procédure Reductions

Procédure utilisant

- OpFraction_Puiss
- Transforme_Inconnue_en_Nombre
- Supprime_Noeud

Algorithme

Initialisation de Ptr1, Ptr2, Ptr3.
Si la nature des deux éléments est (< Nbre > pour < Valeur >) et (< Nbre > ou < Inc > pour < Facteur>)
Alors, Nous élevons le premier élément à une puissance égale à la valeur du second
Si <Facteur> est une inconnue
Alors nous faisons la somme des exposants
Si la somme des exposants est égale à 0
Alors < Facteur> est un nombre
Si la valeur de l'exposant est < > de 1 et < > de -1
Alors nous envoyons un message d'erreur
Supprime_Noeud (ptr1)
dispose (ptr2)
Arbre_Syntaxique ← ptr3
Sinon, nous construisons un nouvel arbre dont la racine est le symbole teminal <Puiss>,
<Facteur> devient le sous-arbre gauche et <Valeur>, le sous-arbre droit
Arbre_Syntaxique^.Categorie_Evol ← Facteur

Procedure Reduct_8 *{ <Terme> ::= <Terme> / <Facteur> }*

Variables globales

Msg	: PChar	% Message de guidage pour l'utilisateur
Erreur	: Boolean	% Indicateur utilisé pour signaler que la syntaxe du string *Equ_dollar* est incorrecte lorsqu'il est initialisé à *False*.
Arbre_Syntaxique	: PArbre	% Pile en cours de construction représentant un membre de l'équation. Elle contient les noeuds déjà analysés lexicographiquement et chaînés.

Variables intermédiaires

Ptr1	: PArbre	% Pointeur vers l'élément N_n placé au sommet de la pile *Arbre_Syntaxique*.
Ptr2	: PArbre	% Pointeur vers l'avant-dernier élément (N_{n-1}) placé au sommet de la pile *Arbre_Syntaxique*.
Ptr3	: Parbre;	% Pointeur vers l'élément (N_{n-2}) de la pile *Arbre_Syntaxique*.
Autre_Cas	: Boolean;	% Indicateur boolean
Simul_Degre	: integer;	% Variable « bidon » nécessaire si nous souhaitons utiliser la procedure 'Inverse_Fraction' pour <Nbre>

Effet

Cette procédure crée un nouvel arbre de type <Terme> (catégorie syntaxique) dont les attributs seront déterminés en fonction de la nature des deux éléments <Terme> et <Facteur>.

Préconditions

Erreur est mis à faux
Arbre_Syntaxique, *Arbre_Syntaxique*^.suivant, *Arbre_Syntaxique*^.suivant^.suivant sont non vides

Postcondition

Arbre_Syntaxique' a été modifié, *Arbre_Syntaxique^.Categorie_Evol'* = Terme

Exceptions :

(1) Si l'expression contenant l'inconnue est élevée à une puissance différente de -1, 0 et de 1.
(2) Si la procédure OpFraction_Divis renvoie un message d'erreur à l'utilisateur.
(3) Si la procédure Inverse_Fraction renvoie un message d'erreur à l'utilisateur.
Msg contient un message de guidage pour l'utilisateur
Erreur est mis à vrai

Procédure utilisée par la procédure Reductions

Procédure utilisant

- OpFraction_Divis
- Inverse_Fraction
- Supprime_Noeud

Algorithme

Initialisation de Ptr1, Ptr2, Ptr3.
Simul_degre ← faux

- **Si** la nature des deux éléments est (< Nbre > pour < Facteur >) et (< Nbre > ou < Inc > pour < Terme>)
  Alors, nous effectuons le produit de ces deux éléments
  Supprime_Noeud (ptr1)
  dispose(Ptr2)
  Arbre_Syntaxique ← ptr3
  Sinon Autre_Cas ← vrai
- **Si** la nature des deux éléments est <Inc>
  Alors, nous effectuons le produit des coefficients et la différence des exposants
  Si la différence des exposants est égale à 0
  Alors < Terme> est un nombre
  Si la valeur de l'exposant est < > de 1 et < > de -1
  Alors nous envoyons un message d'erreur
  Supprime_Noeud (ptr1)
  dispose (ptr2)
  Arbre_Syntaxique ← ptr3
  Sinon Autre_Cas ← vrai
- **Si** la nature des deux éléments est (<Nbre> pour <Terme>) et (<Inc> pour <Facteur>)
  Alors, nous effectuons le produit de ces deux éléments et multiplions l'exposant de l'expression contenant l'inconnue par -1
  Dispose(ptr2)
  Supprime_Noeud (Ptr3);
  Arbre_Syntaxique ← Ptr1;
  Sinon Autre_Cas ← vrai
- **Si** Autre_Cas est mis à vrai
  Alors **Si** la nature de <Facteur> est <Inc> ou <Nbre>
  Alors nous inversons la fraction représentant le nombre (<Nbre>) ou le coefficient de l'expression contenant l'inconnue (<Inc>)
  Sinon, nous construisons un nouvel arbre dont la racine est le symbole teminal <Divis>, <Terme> devient le sous-arbre gauche et <Facteur>, le sous-arbre droit

Arbre_Syntaxique^.Categorie_Evol ← Terme

Procedure Analyse_Membre

(Equ_dollar : string; var Erreur : boolean; Num_Mbre : integer; var Msg : PChar; var i : integer; var Arbre_Syntaxique : PArbre);

Arguments

Equ_dollar	: String	% Expression mathématique introduite par l'utilisateur à laquelle nous avons ajouté le caractère '$' pour en signaler la fin.
Num_Membre	: Integer	% Indicateur permettant de connaître le membre de l'équation qui est en cours d'analyse.

Résultats

Msg	: PChar	% Message de guidage pour l'utilisateur
Erreur	: Boolean	% Indicateur utilisé pour signaler que la syntaxe du string *Equ_dollar* est incorrecte lorsqu'il est initialisé à *False*.
i	: Integer	% Indice du string *Equ_dollar*.
Arbre_Syntaxique	: PArbre	% Pile en cours de construction représentant un membre de l'équation. Elle contient les noeuds déjà analysés lexicographiquement et chaînés.

Variables intermédiaires

Arbre_Lexico	: PArbre	% Pile composée de noeuds, constituée essentiellement de symboles terminaux auxquels nous n'avons pas encore essayé d'appliquer une règle syntaxique.
Fin_Membre	: Boolean	% Indicateur de fin d'analyse d'un membre

Constante

```
Table_Decision : array [Membre .. Rien, Inc .. Rien] of TTab_Dec
                 = ((E,E,D,D,E,E,E,E,D,R),
                    (E,E,R,R,D,D,E,E,R,R),
                    (E,E,R,R,R,R,D,E,R,R),
                    (E,E,R,R,R,R,R,E,R,R),
                    (E,E,R,R,R,R,R,E,R,R),
                    (A,E,R,R,R,R,R,E,R,R),
                    (D,D,E,D,E,E,E,D,E,E),
                    (D,D,E,D,E,E,E,D,E,E),
                    (D,D,E,D,E,E,E,D,E,E),
                    (D,D,E,D,E,E,E,D,E,E),
                    (P,D,E,D,E,E,E,D,E,E),
                    (D,D,E,D,E,E,E,D,E,E),
                    (E,E,R,R,R,R,R,E,R,R),
                    (D,D,E,D,E,E,E,D,E,E));
```

% Table de décision (cfr. 5.2.2.3 La table de décision, page 87)
Les colonnes représentent les éléments de la pile *Arbre_Lexico* tandis que les lignes, les éléments de la pile Arbre_Syntaxique.

Effet

Transforme une partie de l'expression mathématique introduite par l'utilisateur, de sa représentation sous forme de caractères vers sa représentation sous forme d'arbre algébrique.

Préconditions

Le dernier élément de Equ_dollar est le caractère '$'
Symbole = Equ_dollar[i-1]
0 < *i* < = Length (Equ_dollar) +1
Num_Membre = 1 ou *Num_Membre* = 2
Msg = ''
Erreur est mis à faux

Postconditions

0 < *i'* < = i+1 < = Length (Equ_dollar)
Symbole' = Equ_dollar [i'-1]
Arbre_syntaxique' a été modifié, *Arbre_Syntaxique^.Categorie_Evol'* = Membre

Exceptions :

(1) Si la procédure *AnalLex_et_Deplace_Objet* renvoie un message d'erreur à l'utilisateur
(2) Si la procédure *Reductions* renvoie un message d'erreur à l'utilisateur
(3) Table_Decision [Arbre_Syntaxique^.Categorie_Evol,
Arbre_Lexico^.Categorie_Evol] = E
(4) Si Arbre_Syntaxique^.suivant^.Categorie_Evol < > Rien et
Si Arbre_Syntaxique^.suivant^.Categorie_Evol = ParG

Msg contient un message de guidage pour l'utilisateur
Erreur est mis à vrai

Procédure utilisée par la procédure Analyseur_Syntaxique

Procédure utilisant

- Char_Suivant
- AnalLex_et_Deplace_Objet
- Reductions
- Puissance_Inconnue
- Ajoute_Fois

Algorithme

Fin_Membre ← faux
Créer Arbre_Lexico et Arbre_Syntaxique
Initialisation des attributs des deux piles
Tant que Arbre_Syntaxique^.Categorie_Evol = Rien et il n'y a pas encore d'erreur syntaxique
et que nous n'avons pas encore fini d'analyser un membre de l'équation
AnalLex_et_Deplace_Objet(Symbole,Msg,Erreur,Arbre_Syntaxique,
Arbre_Lexico,Fin_Membre);
Répéter
Si Table_Decision [Arbre_Syntaxique^.Categorie_Evol, Arbre_Lexico^.Categorie_Evol] est
A : Ajoute_Fois
D : AnalLex_et_Deplace_Objet (Symbole,Msg,Erreur,Arbre_Syntaxique,Arbre_Lexico);
E : Erreur est mis à vrai
P : Puissance_Inconnue (Arbre_Syntaxique^.suivant^.Categorie_Init,Msg);
Erreur est mis à vrai
R : Reductions(Arbre_Syntaxique,Msg,Erreur);
Jusqu'à ce que nous trouvions une erreur ou (Arbre_Lexico^.Categorie_Evol=Rien)
Tant que Arbre_Syntaxique^.Categorie_Evol < > Membre [ou que nous trouvions une erreur]
Reductions(Arbre_Syntaxique,Msg,Erreur)
Si (Arbre_Syntaxique^.suivant^.Categorie_Evol< >Rien
Alors Si Arbre_Syntaxique^.suivant^.Categorie_Evol = ParG
Alors nous envoyons un message d'erreur à l'utilisateur

5.3 L'unité « Canon »

L'unité « Canon » regroupe l'ensemble des procédures qui ont été utilisées pour mettre l'équation, c'est-à-dire l'expression mathématique introduite par l'utilisateur, sous sa forme canonique. A partir de cette équation canonique, nous pourrons également donner la solution de l'équation.

La conception

Une expression peut être représentée sous différentes formes. Pour l'analyse syntaxique, nous avions choisi la plus usuelle, la notation infixée. Par exemple, la somme de a et de b s'écrit « a + b ». Il existe deux autres possibilités de placer l'opérateur par rapport aux deux opérandes. Dans la première, l'opérateur précède les deux opérandes, il s'agit de la notation préfixée « + a b » ou notation polonaise. Dans la seconde, l'opérateur suit les deux opérandes, on parle de notation postfixée « a b + » ou de notation polonaise inverse. Pour mettre notre équation sous sa forme canonique et la résoudre, nous avons choisi d'utiliser la notation postfixée.

Préalablement, nous devons transformer l'expression mise sous forme infixée lors de l'analyse syntaxique (représentation arborescente) à la forme postfixée (pile). Nous avons choisi d'utiliser une pile pour regrouper sous trois catégories seulement les éléments de l'arbre syntaxique : les valeurs de l'expression avec l'inconnue, les nombres et les opérateurs. La pile *p* est construite et contient donc des éléments de trois natures différentes dont le traitement s'effectue de la manière suivante :

1. Si l'élément est un opérateur, il est placé sur la pile d'évaluation *pilev*
2. Si l'élément est un nombre ou une expession avec l'inconnue, il faut agir en fonction de l'élément précédent,
   - ♦ S'il s'agit d'un opérateur, le nombre ou l'expression avec l'inconnue est simplement placé sur *pilev;*
   - ♦ S'il s'agit d'un nombre ou de l'expression avec l'inconnue, il faut l'enlever de *pilev,* puis enlever encore un élément (qui doit être un opérateur) de *pilev,* ensuite effectuer. Le résultat est mis dans *temp* et nous reprenons au point 2.

Remarque : Nous avons besoin d'une autre pile *pilinc* qui contiendra les valeurs de l'expression contenant l'inconnue de l'équation. Si nous sommes en présence d'une somme (ou d'une différénce) d'un nombre et d'une valeur de l'expression de l'inconnue, cette valeur va être ajoutée à la pile *pilinc.* Si l'opérateur est *, / ou ^ et que pilinc n'est pas vide, il faut enlever l'élément au sommet de *pilinc*, effectuer et placer le résultat sur *pilinc.* Si le résultat est un nombre, par exemple le résultat de x ^ 0 est égal à 1, nous ne le replaçons pas sur *pilinc* mais avant de reprendre au point 2, nous l'ajoutons au résultat trouvé *temp.*

Conclusion générale

Nous voilà arrivés au terme de ce travail. Avons-nous atteint les objectifs que nous nous étions fixés?

Nous avons élaboré un programme permettant d'aider un élève et un professeur dans leur tâche mais si nous regardons ce travail plus en profondeur, nous voyons que nous avons réalisé un outil qui permet de manipuler des expressions mathématiques (Vérification d'une syntaxe, etc).

Nous avons choisi de traiter les problèmes du premier degré à une inconnue mais la plupart des procédures et des fonctions (unité syntaxe et unité sémantique) peuvent être « réutilisables » pour des problèmes du second degré, par exemple.

Nous avons choisi un langage orienté-objet. Cependant, nous avons gardé une démarche procédurale dans la rédaction de notre programme (sauf au niveau de l'implémentation des objets visuels).

Nous regrettons d'avoir réduit le contenu de notre analyse logicielle. Mais, l'essentiel de ce travail réside dans les deux chapitres suivants : l'analyse pédagogique et les concepts et les stratégies mis en jeu.

La satisfaction que nous retirons de notre projet ne se borne pas à sa réalisation. A travers ce mémoire, nous avons acquis une certaine confiance en nous, un désir d'aller le plus loin possible. Nous pourrions nous étendre encore et encore sur ce sujet. Il est très difficile de se fixer des limites.

Ce projet n'est pas terminé. Il reste un certain nombre de choses, d'une part, à affiner (l'analyse sémantique, l'implémentation du diagnostiqueur - outil de remédiation,etc) et d'autre part, à créer (l'évaluation de l'exercice de l'élève,etc). Nous espérons que des étudiants des années à venir seront séduits par ce mémoire et qu'ils le poursuivront.

Bibliographie

[ADGP82] ADAM A., CLOSE Ph., GOOSSENS F., TROMME M.,
Coordination G. Halin & F. Lousberg,
« *Mathématisons 34 - Guide pédagodique* »,
Les éditions A. De Boeck, Bruxelles, 1982.

[ADMA82] ADAM A., CLOSE Ph., GOOSSENS F., TROMME M.,
Coordination G. Halin & F. Lousberg,
« *Mathématisons 34 - Manuel* »,
Les éditions A. De Boeck, Bruxelles, 1983 (2ème édition).

[ARMI86] ARMICI Jean-Claude,
« *Programmer en Turbo Pascal : Notions avancées* »,
Editions Librairie Informatique, Genève, 1986.

[AUSU68] AUSUBEL,
« *Educational Psychology. A cognitive view* »,
New York (1968).

[BAET94] BAETMANS A., DELEUZE-RONVAUX M., HAUBRUGE-DESTREE, M-Cl., POISSEROUX G., SMITS J.,
Coordination B. Gilot, dirigé par J. Masset,
Collection « Mathbase » (livret H) - Calcul numérique et algébrique, géométrie, graphiques,
Editions Erasme, Namur, 1994 (6ème édition).

[BIP56] FARINE Avigdor,
« *La fée informatique ne changera pas l'élève en génie en herbre* »,
Article paru dans « BIP BIP - Informations et réflexions sur l'informatique en éducation »,
Juin 1990, N°56.

[BRAS65] M. BRASSEUR, J. COHEN,
« *Algorithmes d'analyse syntaxique pour langages context-free* »,
R.F.T.I. - CHIFFRES, vol.8, N°2, 1965, pages 95-119
& N°3, 1965, pages 195-213.

[BRUL93] BRULL-GUILLAUME Joëlle, DAVID-GILBERT,
Sous la direction de J. DONNAY,
« *Les nouveaux rôles de l'enseignant impliqué dans un processus d'innovation pédagogique* »,,
Rapport de recherche en éducation, N°36/92, Octobre 1993.

[BOUT82] BOUTRIAU J., PATERNOTTRE A.,
« *Savoir et savoir faire en mathématique, 3ème T* »,
Edition H. Dessain, Liège, 1982.

[DEPO87] DEPOVER Christian,
« L'ordinateur média d'enseignement - un cadre conceptuel »,
Pédagogies en développement - Problématiques et recherches,
Les éditions De Boeck, 1987.

[DONN95] DONNAY Jean,
« Syllabus du cours de pédagogie générale, de méthodologie générale et de pédagogie expérimentale »,
Année académique 1994-1995.

[JOUR94] JOURNET Nicolas,
« La mauvaise note » ,
Article paru dans le magazine « Science et vie »,
Septembre 1994, n°188.

[LANT96] LANTIM Dick,
« Delphi - Programmation avancée »,
Editions Eyrolles, Paris, 1996.

[LECO93] LECOMTE Jacques,
« Les mécanismes de l'apprentissage »,
Article paru dans le magazine « Sciences Humaines »,
Octobre 1993, n°32.

[LERO94] H. LEROY,
« Syllabus du cours de calculabilité et complexité »,
Année académique 1993-1994.

[PAIR64] C. PAIR,
« Arbres, piles et compilation »,
R.F.T.I. - CHIFFRES, vol.7, N°3, 1964, pages 199-216.

[REEV67] C. M. REEVES,
« *Description of a syntax-directed translator* »,
Computer Journal, vol.10, N°3, novembre 1967, pages 244-255.

[SCHO62] SCHONS N.-J.,
« Eléments d'algèbre »
Editon « La procure », Namur, 1962 (7ème édition).

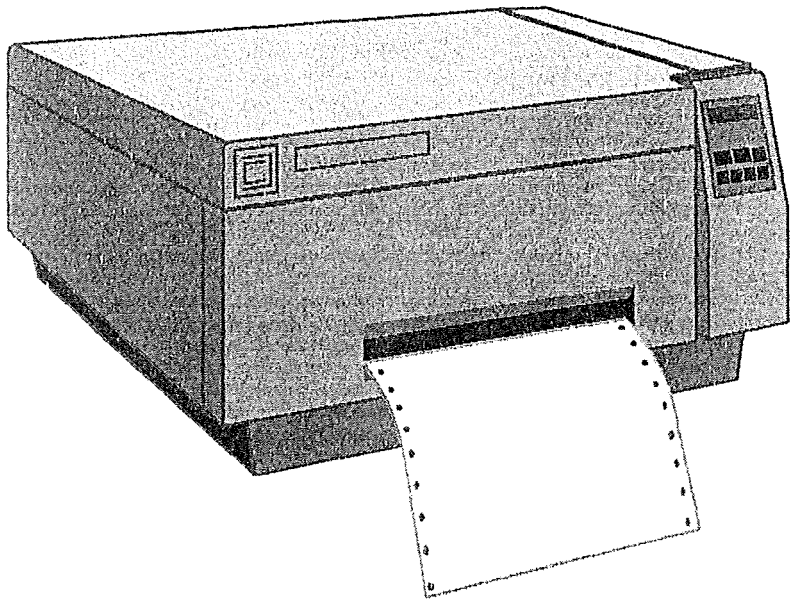
[TARD92] TARDIF Jacques,
« *Pour un enseignement stratégique - L'apport de la psychologie cognitive* »,
Les Editions Logiques, 1992.

[VAND92] VANDERDONCKT Jean,
« Corpus ergonomique minimal des applications de gestion »,
Rapport IHM/Règles/10,
29 octobre 1992.

[WATS95] WATSON Deryn et TINSLEY David,
« *Integrating Information Technology into Education* »,
Editions Chapman et Hall, 1995.

Annexe 1

Le listing



```
{*****************************************************************************}
{*                              UNIT FRACTION                                *}
{*                                                                           *}
{* Cette unité regroupe toutes les procédures relatives aux traitements     *}
{* d'expressions contenant des fractions :                                   *}
{* - Simplification d'une fraction                                           *}
{* - Inversion du numérateur avec le dénominateur                            *}
{* - Addition, Multiplication, Division de fractions                         *}
{* - Fraction élevée à une puissance                                         *}
{*****************************************************************************}

UNIT Fraction;

{ ************************************************************************** }
{ ********************* INTERFACE ****************************************** }
{ ************************************************************************** }


TERFACE

{ ************************** Déclaration des types ************************** }

pe
   TFraction = record
                    numerateur   : real;
                    denominateur : real;
               end;

   TCategorie = (Membre,Terme,Facteur,Valeur,Inc,Nbre,Plus,Moins,Fois,Divis,
                 Puiss,ParG,ParD,Rien);
   { Ensemble des catégories syntaxiques de notre grammaire}


{ ************************** Déclaration des entêtes ************************** }

ocedure Simplifie_Fraction (var fraction : TFraction);
Procedure Inverse_Fraction (var fraction : TFraction; Categorie : TCategorie;
                            var degre    : integer; var Erreur : boolean;
                            var Msg      : PChar    );
ocedure OpFraction_Plus (var fraction1 : TFraction;fraction2 : TFraction);
Procedure OpFraction_Fois (var fraction1 : TFraction;fraction2 : TFraction);
Procedure OpFraction_Divis(var fraction1 : TFraction;fraction2 : Tfraction;
                           var Msg       : Pchar; var Erreur : boolean );
ocedure OpFraction_Puiss(var fraction1 : TFraction;fraction2 : Tfraction;
                           var Msg : PChar; var Erreur : boolean );

{ ************************************************************************** }
{ ********************* IMPLEMENTATION ************************************* }
{ ************************************************************************** }

PLEMENTATION



ocedure Simplifie_Fraction ( var fraction : TFraction);
{ BUT : Cette procédure permet de simplifier une fraction.
        Elle est basée sur l'algorithme d'Euclide.          }

r nbre1 : real;      { Le numérateur de la fraction   }
  nbre2 : real;      { Le dénominateur de la fraction }
  pgcd  : real;      { Le plus grand commun diviseur des nombres nbre1 et nbre2 }

gin
    with fraction do
      begin
           nbre1:=numerateur;
           nbre2:=denominateur;
           pgcd:=nbre1-nbre2*int(nbre1/nbre2);
           while pgcd <> 0 do
            begin
                nbre1:=nbre2;
                nbre2:=pgcd;
                pgcd:=nbre1-nbre2*int(nbre1/nbre2);
            end;
           numerateur:=numerateur/nbre2;
           denominateur:= denominateur/nbre2;
           if (abs(denominateur)<> denominateur)
              then begin
                        numerateur  :=(-1)*numerateur;
                        denominateur:=(-1)*denominateur
```

```
                    end
        end;
End;
{ ---------------------- Fin Simplification_Fraction ------------------------ }


Procedure Inverse_Fraction  (var fraction : TFraction; Categorie : TCategorie;
                             var degre    : integer; var Erreur : boolean;
                             var Msg : PChar);

{ BUT : Cette procédure échange le contenu du numérateur avec celui du
        dénominateur. }

var Temp : real; { Variable tampon contenant la valeur du numérateur }

Begin
     If fraction.numerateur = 0
        then begin
                  Msg:= 'Tu ne peux pas diviser par zéro !';
                  Erreur := true;
                  Exit
             end
        else begin
                  With Fraction do
                   begin
                        Temp:=numerateur;
                        numerateur:=denominateur;
                        denominateur:=Temp;
                        if (abs(denominateur)<> denominateur)
                           then begin
                                     numerateur  :=(-1)*numerateur;
                                     denominateur:=(-1)*denominateur
                                end;
                   end;
                  if Categorie = Inc then degre := (-1)*degre;
             end
End;
{ ----------------------- Fin Inverse_Fraction ------------------------------ }


Procedure OpFraction_Plus (var fraction1 : TFraction;fraction2 : Tfraction);
{ Somme de fractions : p/q + r/s   = (ps+qr)/qs }

Begin
     fraction1.numerateur :=  (fraction1.numerateur*fraction2.denominateur)
                            + (fraction1.denominateur*fraction2.numerateur);
     fraction1.denominateur :=(fraction2.denominateur*fraction1.denominateur);
     Simplifie_Fraction (fraction1)
End;
{ ---------------------- Fin OpFraction_Plus ------------------------------- }


Procedure OpFraction_Fois (var fraction1 : TFraction;fraction2 : Tfraction);
{ Produit de fractions : (p/q)*(r/s) = pr/qs }

Begin
     fraction1.numerateur := (fraction1.numerateur*fraction2.numerateur);
     fraction1.denominateur :=(fraction2.denominateur*fraction1.denominateur);
     Simplifie_Fraction (fraction1)
End;
{ ---------------------- Fin OpFraction_Fois ------------------------------- }


Procedure OpFraction_Divis (var fraction1 : TFraction;fraction2 : Tfraction;
                            var Msg : Pchar; var Erreur : boolean );
{ Quotient de fractions : (p/q)/(r/s) = ps / qr et r <> 0 }

Begin
     If fraction2.numerateur = 0
        then begin
               Msg:= 'Tu ne peux pas diviser par zéro !';
               Erreur := true;
               Exit
             end
        else begin
               fraction1.numerateur
```

```
                    :=(fraction1.numerateur*fraction2.denominateur);
            fraction1.denominateur
                    :=(fraction2.numerateur*fraction1.denominateur);
            Simplifie_Fraction (fraction1)
          end
ıd;
 ---------------------- Fin OpFraction_Divis ------------------------------}


:ocedure OpFraction_Puiss (var fraction1 : TFraction;fraction2 : Tfraction;
                           var Msg : PChar; var Erreur : boolean );

 Puissance d'une fraction : (p/q) ^ r
 Si p = 0 et r <> 0 alors (p/q) ^ r = 0
 Si r = 0 alors (p/q) ^ r = 1
 Si r<> 0 alors (p/q) ^ r = (p/q)* (p/q) * ... r fois }

ır temp1    : real;    { Variable tampon contenant la valeur du numérateur }
   temp2    : real;    { Variable tampon contenant la valeur du dénominateur }
   exposant : real;    { Compteur indiquant le (nombre de fois -1) que nous
                         avons multiplié fraction1 par fraction2  }

Begin
  If (fraction1.numerateur = 0)
     then begin
          if (fraction2.numerateur = 0)
             then begin
                  Msg := 'Tu ne peux pas élever le nombre zéro '
                        +'à la puissance zéro !';
                  Erreur := true;
                  Exit
                  end
             else fraction1.numerateur := 0
          end
     else begin
          if (fraction2.numerateur = 0)
             then begin
                  fraction1.numerateur := 1;
                  fraction1.denominateur := 1
                  end
             else begin
                  if (fraction2.denominateur <> 1)
                     then begin
                          Msg := 'Dois-tu vraiment calculer une racine ?';
                          Erreur := true;
                          Exit
                          end
                     else begin
                          if (abs(fraction2.numerateur)<> fraction2.numerateur)
                             then begin
                                  temp1 := fraction1.numerateur;
                                  fraction1.numerateur:= fraction1.denominateur;
                                  fraction1.denominateur:=temp1;
                                  fraction2.numerateur:= (-1)*fraction2.numerateur
                                  end;
                          exposant:=1;
                          temp1 := fraction1.numerateur;
                          temp2 := fraction1.denominateur;
                          while (exposant < fraction2.numerateur) do
                            begin
                              fraction1.numerateur:=fraction1.numerateur*temp1;
                              fraction1.denominateur
                                       := fraction1.denominateur*temp2;
                              exposant:=exposant+1
                            end
                          end
                  end
          end;
End;
{ ---------------------- Fin OpFraction_Puiss ------------------------------ }


End.
```

```
{*****************************************************************************}
{*                                UNIT SYNTAXE                               *}
{*                                                                           *}
{* Cette unité regroupe toutes les fonctions et les procédures relatives à   *}
{* l'implémentation de l'analyseur syntaxique  :                             *}
{* - Char_suivant                                                            *}
{* - AnalLex_et_Deplace_Objet                                                *}
{* - Supprime_Noeud                                                          *}
{* - Transforme_Inconnue_en_Nombre                                           *}
{* - Reductions                                                              *}
{* - Analyse Membre                                                          *}
{* - Parcours_Membre                                                         *}
{* - Analyseur_Syntaxique                                                    *}
{*****************************************************************************}

UNIT SYNTAXE;

{*****************************************************************************}
{********************** INTERFACE ********************************************}
{*****************************************************************************}

INTERFACE

uses Fraction, Sysutils;

{************************** Déclaration des types ****************************}

type
    PArbre = ^TArbre;
    TArbre = record
                Categorie_Evol : TCategorie;
                Suivant        : PArbre;
                case Categorie_init : TCategorie of
                     Inc                          :(coefficient : TFraction;
                                                    Degre : Integer);
                     Nbre                         :(Val_Nbre  :TFraction);
                     Plus,Moins,Fois,Divis,Puiss  :(POperand1,
                                                    POperand2:PArbre);
                     ParG,ParD,Rien               :();
             end;
    { Categorie_evol représente la catégorie syntaxique de l'élément.        }
    { Categorie_init représente le symbole terminal d'où provient l'élément. }
    {                Il caractérise chaque noeud de notre arbre.             }
    { Suivant est un pointeur vers l'élément suivant                         }

    Tab_Membres = array [1..2] of PArbre;


{ ********************* Déclaration des variables globales ******************}
Var
    i          : integer;    { Indice de Equ_dollar                          }
    Num_Inc    : integer;    { Code AscII de la lettre représentant l'inconnue }
    Cpt_Inc    : integer;    { Compteur du nombre d'occurences d'un terme
                               contenant l'inconnue                          }
    Symbole    : char;       { Caractère extrait de Equ_dollar, compatible avec
                               un symbole terminal de notre grammaire        }
    inconnue   : char;       { Lettre représentatnt l'inconnue               }
    Equ_dollar : string;     { Expression mathématique introduite par
                               l'utilisateur à laquelle nous avons ajouté le
                               caractère '$' pour en signaler la fin         }


{ ************************** Déclaration des entêtes **************************}

Function Char_Suivant (Equ_dollar : string; var i :integer): char;
Procedure AnalLex_et_Deplace_Objet (Var Symbole : char; Var Msg : Pchar;
          Var Erreur : boolean; Var Arbre_Syntaxique : PArbre;
          Var Arbre_Lexico : PArbre;Num_Mbre : integer);
Procedure Supprime_Noeud (Noeud : PArbre);
Function Transforme_Inconnue_en_Nombre (Noeud_Inc : PArbre): PArbre;
Procedure Reductions ( Var Arbre_Syntaxique : PArbre; Var Msg : PChar;
                       Var Erreur : boolean);
Procedure Analyse_Membre ( Equ_dollar : string; var Erreur : boolean;
                           Num_Mbre : integer; var Msg : PChar;
                           var i : integer; var Arbre_Syntaxique : PArbre);
Procedure Parcours_Membre(Mbre: PArbre; inconnue : char; var Cpt_Inc : integer);
Procedure Analyseur_Syntaxique (var Msg : PChar ; var Erreur : boolean;
                                var arbre_equation : Tab_Membres;
                                    inconnue : char; Equation : string);
```

```
{ ********************************************************************************* }
{ ********************** IMPLEMENTATION ********************************************* }
{ ********************************************************************************* }

IMPLEMENTATION


Function Char_suivant (Equ_dollar : string; var i :integer): char;
  {But : Sélectionne le ième caractère du string Equ_dollar et incrémente
         l'indice i d'une unité.}

Begin
     Char_suivant := Equ_dollar [i];
     i:=i+1
End;
{ ************************ Fin de Char_suivant ****************************** }



Procedure AnalLex_et_Deplace_Objet (Var Symbole :char;
                                    Var Msg : Pchar;
                                    Var Erreur  : boolean;
                                    Var Arbre_Syntaxique : PArbre;
                                    Var Arbre_Lexico  : PArbre;
                                    Num_Mbre : integer);

{ But : Cette procédure anlyse lexicographiquement Equ_dollar et en extrait un
        caractère compatible avec un symbole terminal de notre grammaire.
        Elle ajoute un élément de Arbre_Lexico à Arbre_syntaxique              }

Const
Lettres : set of char =['a'..'z','A'..'Z']; {Ensemble des lettres de l'alphabet}
Chiffres : set of char = ['0'..'9',',','.'];{Ensemble des caractères pouvant
                                             apparaître dans un nombre }

Var
    Bouge_Objet     : boolean;       { Indicateur de déplacement d'un élément de
                                       Arbre_Lexico vers Arbre_Syntaxique       }
    Temp_Categorie : TCategorie;     { Symbole terminal d'où provient l'élément
                                       en cours d'analyse                       }
    Temp_Nbre       : TFraction;     { Valeur numérique de l'élément en cours
                                       d'analyse lexicographique. Le symbole
                                       terminal est un nombre                   }
    Temp_coefficient : TFraction;    { Valeur numérique du coefficient de
                                       l'élément en cours d'analyse
                                       lexicographique. Le symbole terminal est
                                       une expression contenant l'inconnue }
    Ajout_Noeud     : PArbre;        { Variable dynamique temporaire utilisée
                                       lorsque nous ajoutons des éléments à
                                       Arbre_Syntaxique                         }
    Temp_Degre      : integer;       { Valeur numérique de la puissance de
                                       l'élément en cours d'analyse
                                       lexicographique. Le symbole terminal est
                                       une expression contenant l'inconnue }


Procedure Repere_Inconnue;
{ But : Cette procédure crée un élément <Inc>. }

Begin
     If  ((ord(Symbole) = (num_inc+32)) or (ord(Symbole) = num_inc))
       Then begin
                Symbole := Char_suivant(Equ_dollar,i);
                if    (ord(Symbole) = num_inc)
                   or (ord(Symbole) = (num_inc+32))
                   then begin
                              Erreur:=True;
                              Msg:= 'Une seule lettre suffit '
                                          + 'pour représenter l''inconnue.'
                        end
            end
       Else begin
                Erreur:=True;
                Msg:= 'L''inconnue de l''équation est représentée par'
                             + ' un autre caractère.';
            end;
     Bouge_Objet := true;
     Temp_Categorie:=Inc;
     Temp_coefficient.numerateur :=1;
```

```
    Temp_coefficient.denominateur := 1;
    Temp_Degre := 1;
End;
{ ---------------------- Fin Repere_Inconnue ---------------------------------- }


procedure Reforme_Nombre;
{ But : Cette procédure construit un "token" formé de la succession des
        caractères appartenant à Chiffres. Elle teste également l'existence
        d'un double point. Elle crée un élément <Nbre>.                        }

Var
   relie : boolean;       {Indicateur indiquant si le point décimal ou la virgule
                           a déjà été introduite}
   stop  : boolean;       {Indicateur de fin de construction du "token"}
   code  : integer;       {Si le Token est incorrecte, l'indice du caractère
                           posant problème est placé dans Code, sinon Code est
                           initialisé à zéro. }
   stringvalue : string; { Token}

Begin
    relie := false;
    stop:=false;
    stringvalue :='';
    If Symbole = ',' then Symbole := '.';
    If Symbole = '.'
       then
           begin
                relie := true;
                stringvalue :='0'
           end;
    stringvalue := stringvalue + Symbole;
    Symbole := Char_suivant(Equ_dollar,i);
    While not stop and (Symbole in Chiffres)do
      begin
        If (Symbole = ',') or (Symbole = '.')
           then begin
            If relie
              then
                begin
                     Erreur :=true;
                     Msg:= 'Tu as déjà entré le point décimal pour ce nombre.';
                     stop:= true
                end
              else
                begin
                   If Symbole = ',' then Symbole := '.';
                   stringvalue := stringvalue + Symbole;
                   Symbole := Char_suivant(Equ_dollar,i);
                   relie:=true
                end
                end
           else begin
            stringvalue := stringvalue + Symbole;
            Symbole := Char_suivant(Equ_dollar,i)
                end
      end;
    Val(stringvalue,Temp_Nbre.numerateur,code);
    Temp_Nbre.denominateur:=1;
    Simplifie_Fraction (Temp_Nbre);
    Bouge_Objet:=true;
    Temp_Categorie:=Nbre
d;
{ ---------------------- Fin Repere_Inconnue ---------------------------------- }


{ ---------------------- Début AnalLex_et_Deplace_Objet ------------------------}
gin
    If Arbre_Lexico^.Suivant = nil Then
       begin
         Bouge_Objet :=False;
         While not Bouge_Objet do
          begin
              Case Symbole of
                   '+': Begin
                             Bouge_Objet:=true;
                             Temp_Categorie := Plus;
                             Symbole := Char_suivant(Equ_dollar,i)
                        End;
                   '-':Begin
```

```
                    Bouge_Objet:=true;
                    Temp_Categorie := Moins;
                    Symbole := Char_suivant(Equ_dollar,i);
                End;
            '*':Begin
                    Bouge_Objet:=true;
                    Temp_Categorie := Fois;
                    Symbole := Char_suivant(Equ_dollar,i)
                End;
            '/':Begin
                    Bouge_Objet:=true;
                    Temp_Categorie := Divis;
                    Symbole := Char_suivant(Equ_dollar,i)
                End;
            '^':Begin
                    Bouge_Objet:=true;
                    Temp_Categorie := Puiss;
                    Symbole := Char_suivant(Equ_dollar,i)
                End;
        '{','[','(':Begin
                    Bouge_Objet:=true;
                    Temp_Categorie := ParG;
                    Symbole := Char_suivant(Equ_dollar,i)
                End;
        '}',']',')':Begin
                    Bouge_Objet:=true;
                    Temp_Categorie := ParD;
                    Symbole := Char_suivant(Equ_dollar,i)
                End;
            '=': Begin
                    If (Num_Mbre=2)
                      then begin
                             Msg:='L''équation est syntaxiquement'
                                  +' incorrecte. Le signe d''égalité'
                                  +' a déjà été introduit.';
                             Erreur:=true
                           end
                      else if (Arbre_lexico^.Categorie_Evol = Rien)
                           then begin
                                  Msg:= 'Tu ne peux pas commencer '
                                        +'ton équation par le symbole'
                                        +' d''égalité.';
                                  Erreur :=True
                                end
                           else if (Arbre_lexico^.Categorie_Evol
                                    in [Plus .. ParG])
                                then begin
                                       Msg:='Que signifie cette '
                                       +'expression mathématique?';
                                       Erreur:=true
                                     end;
                    Bouge_Objet:=true;
                    Temp_Categorie:=Rien;

                End;
            '$' :Begin
                    If (Num_Mbre = 1)
                       then begin
                              if (arbre_lexico^.Categorie_evol = Rien)
                              then Msg:='Tu n''as pas entré d''équation!'
                              else Msg:= 'L''équation est incomplète.';
                              Erreur:=true
                            end
                       else If (Arbre_Lexico^.Categorie_evol = Rien )
                             then begin
                               Msg:= 'L''équation est incomplète. '
                                    +'Elle n''a pas de second membre.'
                                    +' A toi de jouer !';
                               Erreur:=true
                                  end;
                    Bouge_Objet:=true;
                    Temp_Categorie:=Rien;

                End;
      Else
        If (Symbole in Lettres)
         Then Repere_Inconnue
         Else if (Symbole in Chiffres)
              then Reforme_Nombre
              else if Symbole=' '
```

```
                                    then Symbole := Char_suivant(Equ_dollar,i)
                                    else
                                      begin
                                           Bouge_Objet:=true;
                                           Msg :=STRPCopy (msg,'Ce caractère, '
                                           + Equ_dollar[i-1]+' ne fait pas partie de '
                                           +'l''équation.');
                                           Erreur :=true
                                      end
                End
             end;
           If not Erreur
              then
                  begin
                       Arbre_Lexico^.Suivant := Arbre_Syntaxique;
                       Arbre_Syntaxique:=Arbre_Lexico;
                       new (Arbre_Lexico);
                       With Arbre_Lexico^ do
                         begin
                              Categorie_Init:=Temp_Categorie;
                              Categorie_Evol:=Temp_Categorie;
                              Case Temp_Categorie of
                                   Inc  : begin coefficient := Temp_coefficient;
                                                Degre     := Temp_Degre
                                          end;
                                   Nbre : begin Val_Nbre :=Temp_Nbre end;
                                   Plus,Moins,Fois,Divis,Puiss : begin
                                                POperand1 := nil;
                                                POperand2 := nil
                                                                end
                              End
                         end;
                       Arbre_Lexico^.Suivant :=nil
                  end
          end
{Analyse_Lexico^.suivnat <> nil}
                                    Else
          begin
            Ajout_Noeud:=Arbre_Lexico^.Suivant;
            Arbre_Lexico^.Suivant:= Arbre_Syntaxique;
            Arbre_Syntaxique:=Arbre_Lexico;
            Arbre_Lexico:=Ajout_Noeud
          end

End;
{ ---------------------- Fin AnalLex_et_Deplace_Objet ------------------------- }


Procedure Supprime_Noeud (Noeud : PArbre);
  But : Cette procédure récursive libère de la place mémoire occupée par Noeud.}

Begin
     Case Noeud^.Categorie_Init of
          Plus,Moins,Fois,Divis,Puiss : begin
                                          Supprime_Noeud(Noeud^.POperand1);
                                          Supprime_Noeud(Noeud^.POperand2)
                                        end;
     End;
     If Noeud <> nil then dispose (Noeud);
End;
{ ---------------------- Fin Supprime_Noeud ----------------------------------- }


Function Transforme_Inconnue_en_Nombre (Noeud_Inc : PArbre): PArbre;
{ But : Cette fonction revoie un élément de type <Nbre> dont la valeur est égale
        au coefficient de l'élément Noeu_Incde type <Inc>.
        Elle supprime l'élément Noeud_inc                                       }

Var Noeud_Nbre : PArbre; {Variable tampon contenant l'élément de type <Nbre>    }

Begin
     new (Noeud_Nbre);
     With Noeud_Nbre^ do
       begin
            Categorie_Evol := Noeud_Inc^.Categorie_Evol;
            Categorie_Init := Nbre;
            Val_Nbre := Noeud_Inc^.coefficient;
            Suivant := Noeud_Inc^.suivant
```

```
      end;
    Supprime_Noeud (Noeud_Inc);
    Transforme_Inconnue_en_Nombre := Noeud_Nbre;
End;
{ -------------------- Fin Transforme_inconnue_en_Nombre -------------------- }


Procedure Reductions ( Var Arbre_Syntaxique : PArbre; Var Msg : PChar;
                       Var Erreur : boolean );

{ But : Cette procédure applique la règle syntaxique Recherche_Regle aux
        éléments formant le sommet de la pile Arbre_Syntaxique.            }


Function Recherche_Regle : integer;

Var
   Categorie_Prec :TCategorie;     {Catégorie syntaxique}
   numero:integer;                 {Numéro identifiant la règle syntaxique que
                                    nous pouvons appliquer aux éléments formant
                                    le sommet de la pile Arbre_Syntaxique.     }

Begin
    numero :=0;
    Case Arbre_Syntaxique^.Categorie_Evol of

      Inc   :begin
               if (Arbre_Syntaxique^.suivant^.Categorie_Evol=Moins)
                  then begin
                       Categorie_Prec
                       := Arbre_Syntaxique^.suivant^.suivant^.Categorie_evol;
                       if (Categorie_Prec=Plus) or (Categorie_Prec=Moins)
                       or (Categorie_Prec=Fois) or (Categorie_Prec=Divis)
                       or (Categorie_Prec=Puiss)or (Categorie_Prec=ParG)
                       or (Categorie_Prec=Rien) then numero:= 1
                       end;
               if (numero = 0) then numero:=2;
            end;


     Nbre   :begin
               if (Arbre_Syntaxique^.suivant^.Categorie_Evol=Moins)
                  then begin
                       Categorie_Prec
                       := Arbre_Syntaxique^.suivant^.suivant^.Categorie_evol;
                       if (Categorie_Prec=Plus) or (Categorie_Prec=Moins)
                       or (Categorie_Prec=Fois) or (Categorie_Prec=Divis)
                       or (Categorie_Prec=Puiss)or (Categorie_Prec=ParG)
                       or (Categorie_Prec=Rien) then numero:= 3
                       end;
               if (numero = 0) then numero:=4;
            end;


     Valeur :begin
               if (Arbre_Syntaxique^.suivant^.suivant^.Categorie_Evol=Facteur)
               and(Arbre_Syntaxique^.suivant^.Categorie_Evol=Puiss)
               then numero:= 5
               else numero:= 6
             end;

     Facteur :begin
               if (Arbre_Syntaxique^.suivant^.suivant^.Categorie_Evol=Terme)
                  then  case Arbre_Syntaxique^.suivant^.Categorie_evol of
                             Fois  : numero := 7;
                             Divis : numero := 8
                        end;
               if (numero = 0) then numero:=9;
             end;

     Terme:begin
               if (Arbre_Syntaxique^.suivant^.suivant^.Categorie_Evol=Membre)
                   and ((Arbre_Syntaxique^.suivant^.Categorie_Evol=Plus)
                    or (Arbre_Syntaxique^.suivant^.Categorie_Evol=Moins))
                   then numero:=10;
               if (numero = 0) then numero:=11
            end;

     ParD:begin
```

```
          If  (Arbre_Syntaxique^.suivant^.suivant^.Categorie_Evol=ParG)
          and (Arbre_Syntaxique^.suivant^.Categorie_Evol=Membre)
           Then begin
           if (Arbre_Syntaxique^.suivant^.suivant^.suivant^.Categorie_Evol=Moins)
             then begin
               Categorie_Prec
               :=Arbre_Syntaxique^.suivant^.suivant^.suivant^.suivant^.Categorie_Evol;
                  if (Categorie_Prec=Plus) or (Categorie_Prec=Moins)
                  or (Categorie_Prec=Fois) or (Categorie_Prec=Divis)
                  or (Categorie_Prec=Puiss)or (Categorie_Prec=ParG)
                  or (Categorie_Prec=Rien) then numero:=12
                  end;
             if (numero = 0) then numero:=13;
                end
           Else begin
             Erreur :=true;
             Msg:='As-tu mis tes parenthèses correctement ?'
                end;
         end;
    End;
    Recherche_Regle:=numero;
nd;
 ------------------- Fin de Recherche_Regle ------------------------------- }


****************** Début des procédures Reduct_i ***********************}


Procedure Reduct_1;        { <Valeur> ::= - <Inc> }

ar Ptr1 : PArbre; {Pointeur temporaire}

Begin
    Ptr1 :=Arbre_Syntaxique^.suivant;
    Arbre_Syntaxique^.suivant := Ptr1^.suivant;
    Arbre_Syntaxique^.Coefficient.numerateur
                := (-1)* Arbre_Syntaxique^.Coefficient.numerateur;
    dispose(Ptr1);
    Arbre_Syntaxique^.Categorie_Evol := Valeur
nd;
{ ------------------------- Fin Reduct_1 ---------------------------------- }


rocedure Reduct_3;        { <Valeur> ::= - <Nbre> }

Var Ptr1 : PArbre;  {Pointeur temporaire}

egin
    Ptr1 :=Arbre_Syntaxique^.suivant;
    Arbre_Syntaxique^.suivant := Ptr1^.suivant;
    Arbre_Syntaxique^.Val_Nbre.numerateur
                := (-1)* Arbre_Syntaxique^.Val_Nbre.numerateur;
    dispose(Ptr1);
    Arbre_Syntaxique^.Categorie_Evol := Valeur
nd;
 ------------------------- Fin Reduct_3 ---------------------------------- }


rocedure Reduct_5;        { <Facteur> ::= <Facteur> ^ <Valeur> }

ar  Ptr1, Ptr2, Ptr3: PArbre; {Pointeurs temporaires}

egin
    Ptr1 := Arbre_Syntaxique;
    Ptr2 := Arbre_Syntaxique^.suivant;
    Ptr3 := Arbre_Syntaxique^.suivant^.suivant;
    { <Nbre> ^ <Nbre>  ou  <Inc> ^ <Nbre> }
    If ((Ptr1^.Categorie_Init = Nbre)and (Ptr3^.Categorie_Init = Nbre))
    or ((Ptr1^.Categorie_Init = Nbre)and (Ptr3^.Categorie_Init = Inc))
       then begin
              { <Nbre> ^ <Nbre> }
              if     ((Ptr1^.Categorie_Init = Nbre)
                 and (Ptr3^.Categorie_Init = Nbre))
                 then begin
                        OpFraction_Puiss(Ptr3^.Val_Nbre, Ptr1^.Val_Nbre,
                                         Msg,Erreur);
                        If Erreur then Exit
                      end
                 { <Inc> ^ <Nbre> }
                 else begin
```

```
                        OpFraction_Puiss(Ptr3^.coefficient, Ptr1^.Val_Nbre,
                                         Msg,Erreur);
                        If Erreur then Exit;
                        Ptr3^.Degre
                            :=(Ptr3^.degre *(Trunc(Ptr1^.Val_Nbre.numerateur)))
                           div (Trunc(Ptr1^.Val_Nbre.denominateur));
                        if (ptr3^.degre = 0)
                           then begin
                                  ptr3 := (Transforme_Inconnue_en_Nombre(ptr3));
                                end
                           else if ((ptr3^.degre <> 1)
                                 and (ptr3^.degre <> -1))
                                then begin
                                       Msg := 'Nous sommes en'
                                       +' train de résoudre un problème du '
                                       +'premier degré(ax+b =0). Essaie de'
                                       +' ne pas élever l''inconnue à une '
                                       +'puissance différente de -1,0 ou de 1.';
                                       Erreur := true;
                                       Exit
                                     end
                     end;
              Supprime_Noeud (ptr1);
              dispose (ptr2);
              Arbre_Syntaxique := ptr3
           end
        { Autres Cas }
        else begin
                   Ptr2^.suivant  := Ptr3^.suivant;
                   Ptr2^.POperand2 :=Ptr1;
                   Ptr2^.POperand1 :=Ptr3;
                   Ptr1^.suivant  := nil;
                   Ptr3^.suivant  := nil;
                   Arbre_Syntaxique := Ptr2
           end;
     Arbre_Syntaxique^.Categorie_Evol := Facteur
End;
{ ------------------------- Fin Reduct_5 ---------------------------------- }


Procedure Reduct_7;        { <Terme> ::= <Terme> * <Facteur> }

ar  Ptr1, Ptr2, Ptr3: PArbre;  {Pointeurs temporaires}
    Autre_Cas       : boolean; {Indicateur pour les cas où <Terme> et <Facteur>
                                ne sont pas des nombres où des expressions
                                contenant l'inconnue                          }
gin
    Ptr1 := Arbre_Syntaxique;
    Ptr2 := Arbre_Syntaxique^.suivant;
    Ptr3 := Arbre_Syntaxique^.suivant^.suivant;
    Autre_Cas  := false;
    If (Ptr1^.Categorie_Init = Nbre)
       then begin
             { < Nbre> * <Nbre> ou <Inc> * <Nbre> }
             if ((Ptr3^.Categorie_Init = Nbre) or (Ptr3^.Categorie_Init = Inc))
                 then begin
                        if (Ptr3^.Categorie_Init = Nbre)
                          then OpFraction_Fois(Ptr3^.Val_Nbre,Ptr1^.Val_nbre)
                          else OpFraction_Fois(Ptr3^.coefficient,Ptr1^.Val_Nbre);
                        Supprime_Noeud(ptr1);
                        dispose(Ptr2);
                        Arbre_Syntaxique := ptr3
                      end
                 else Autre_Cas := true
          end
       else begin
              If (Ptr1^.Categorie_Init = Inc)
                 then begin
                      { <Nbre> * <Inc>  ou  <Inc> * <Inc> }
                      if ((Ptr3^.Categorie_Init = Nbre) or
                          (Ptr3^.Categorie_Init = Inc))
                         then begin
                              If (Ptr3^.Categorie_Init = Nbre)
                                 then OpFraction_Fois(Ptr1^.coefficient,
                                                      Ptr3^.Val_Nbre)
                                 else begin
                                      {<Inc> * <Inc> }
                                        OpFraction_Fois
                                        (Ptr1^.coefficient, Ptr3^.coefficient);
                                         Ptr1^.Degre
```

```
                                        := Ptr1^.Degre + Ptr3^.Degre;
                                    if (ptr1^.degre = 0)
                                    then ptr1 :=
                                    (Transforme_Inconnue_en_Nombre(ptr1))
                                    else if ((ptr1^.degre <> 1)
                                         and (ptr1^.degre <> -1))
                                         then begin
                                   Msg := 'Nous sommes en'
                                   +' train de résoudre un problème du '
                                   +'premier degré(ax+b =0). Essaie de'
                                   +' ne pas multiplier l''inconnue par'
                                   +' elle-même.';
                                   Erreur := true;
                                   Exit
                                              end
                                 end;
                            Arbre_Syntaxique := Ptr1;
                            Ptr1^.suivant := Ptr3^.suivant;
                            dispose(Ptr2);
                            Supprime_Noeud(ptr3);
                            end
                       else Autre_Cas := true;
                   end
               else Autre_Cas := true;
          end;
    { Autres cas}
    If Autre_Cas = true
       then begin
                 Ptr2^.suivant  := Ptr3^.suivant;
                 Ptr2^.POperand2 :=Ptr1;
                 Ptr2^.POperand1 :=Ptr3;
                 Ptr1^.suivant  := nil;
                 Ptr3^.suivant  := nil;
                 Arbre_Syntaxique := Ptr2
          end;
    Arbre_Syntaxique^.Categorie_Evol := Terme
End;
{ ------------------------- Fin Reduct_7 ---------------------------------- }


Procedure Reduct_8;        { <Terme> ::= <Terme> / <Facteur> }

ar  Ptr1, Ptr2, Ptr3: PArbre;   {Pointeurs temporaires}
    Autre_Cas       : boolean;  {Indicateur pour les cas où <Terme> et<Facteur>
                                 ne sont pas des nombres où des expressions
                                 contenant l'inconnue                          }
    Simul_Degre     : integer;  { Variable "bidon" nécessaire si nous
                                  souhaitons utiliser la procedure
                                  Inverse_fraction pour <Nbre>                  }

egin
    Ptr1 := Arbre_Syntaxique;
    Ptr2 := Arbre_Syntaxique^.suivant;
    Ptr3 := Arbre_Syntaxique^.suivant^.suivant;
    Autre_Cas  := false;
    If (Ptr1^.Categorie_Init = Nbre)
       then begin
              { <Nbre> / <Nbre> ou <Inc> / <Nbre> }
              if ((Ptr3^.Categorie_Init = Nbre) or (Ptr3^.Categorie_Init = Inc))
                 then begin
                        if (Ptr3^.Categorie_Init = Nbre)
                           then OpFraction_Divis(Ptr3^.Val_Nbre,Ptr1^.Val_nbre,
                                                 Msg, Erreur)
                           else OpFraction_Divis(Ptr3^.coefficient,Ptr1^.Val_Nbre,
                                                 Msg, Erreur);
                        If Erreur then Exit;
                        Supprime_Noeud(ptr1);
                        dispose(Ptr2);
                        Arbre_Syntaxique := ptr3
                      end
                 else Autre_Cas := true
          end
       else begin
              If (Ptr1^.Categorie_Init = Inc)
                 then begin
                        { <Inc> / <Inc>}
                        if (Ptr3^.Categorie_Init = Inc)
                           then begin
                                  OpFraction_Divis(Ptr3^.coefficient,
                                  Ptr1^.coefficient,Msg,Erreur);
```

```
                                    If Erreur then Exit;
                                    Ptr3^.Degre := Ptr3^.Degre - Ptr1^.Degre;
                                    If (ptr3^.degre = 0)
                                       then ptr3 :=
                                           (Transforme_Inconnue_en_Nombre(ptr3))
                                       else if ((ptr3^.degre <> 1)
                                             or (ptr3^.degre <> -1))
                                             then begin
                                             Msg := 'Nous sommes en'
                                             +' train de résoudre un problème du '
                                             +'premier degré(ax+b =0). Essaie de'
                                             +' ne pas diviser l''inconnue par'
                                             +' elle-même.';
                                             Erreur := true;
                                             Exit
                                                  end;
                                    Supprime_Noeud(ptr1);
                                    dispose(Ptr2);
                                    Arbre_Syntaxique := ptr3
                                 end
                              else begin
                                    { <Nbre> / <Inc> }
                                    if (Ptr3^.Categorie_Init = Nbre)
                                       then begin
                                              OpFraction_Divis
                                              (Ptr3^.Val_Nbre,Ptr1^.coefficient,
                                               Msg,Erreur);
                                              If Erreur then Exit;
                                              Ptr1^.coefficient := Ptr3^.Val_Nbre;
                                              Ptr1^.Degre := (-1)*Ptr1^.Degre;
                                              Ptr1^.Suivant := Ptr3^.suivant;
                                              Dispose(ptr2);
                                              Supprime_Noeud (Ptr3);
                                              Arbre_Syntaxique:=Ptr1;
                                           end
                                       else Autre_Cas := true
                                 end
                        end
                  else Autre_Cas := true
            end;
      {Autres cas}
      If Autre_Cas = true
         then begin
               if ((Ptr1^.Categorie_Init = Nbre) or (Ptr1^.Categorie_Init = Inc))
                   then begin if (Ptr1^.Categorie_Init = Nbre)
                                 then Inverse_Fraction (Ptr1^.Val_Nbre,
                                      Nbre,Simul_Degre,Erreur,Msg)
                                 else Inverse_Fraction (Ptr1^.coefficient,
                                      Inc,Ptr1^.Degre,Erreur,Msg);
                              Ptr2^.Categorie_Init := Fois
                        end;
               Ptr2^.suivant  := Ptr3^.suivant;
               Ptr2^.POperand2 :=Ptr1;
               Ptr2^.POperand1 :=Ptr3;
               Ptr1^.suivant  := nil;
               Ptr3^.suivant  := nil;
               Arbre_Syntaxique := Ptr2
            end;
      Arbre_Syntaxique^.Categorie_Evol := Terme
id;
 ------------------------- Fin Reduct_8 ------------------------------------ }


ocedure Reduct_10;        { <Membre> ::= <Membre> <addop> <Terme> }

Var  Ptr1, Ptr2, Ptr3 : PArbre; {Pointeurs temporaires}

gin
    Ptr1 := Arbre_Syntaxique;
    Ptr2 := Arbre_Syntaxique^.suivant;
    Ptr3 := Arbre_Syntaxique^.suivant^.suivant;
    If (Ptr2^.Categorie_Evol = Moins)
       then begin
              { - <Inc> ou - <Nbre> }
              If((Ptr1^.Categorie_Init = Nbre)or(Ptr1^.Categorie_Init = Inc))
                then begin
                           If(Ptr1^.Categorie_Init = Inc)
                             then Ptr1^.coefficient.Numerateur
                                       := (-1)*Ptr1^.Val_Nbre.Numerateur
                             else Ptr1^.Val_Nbre.Numerateur
```

```
                                    Ptr2^.Categorie_Init := Plus
                            end;
                end;
        { <Nbre> + <Nbre> ou <Inc> + <Inc> }
        If (((Ptr1^.Categorie_Init = Nbre) and (Ptr3^.Categorie_Init = Nbre))
        or  ((Ptr1^.Categorie_Init = Inc) and (Ptr3^.Categorie_Init = Inc) and
             (Ptr3^.Degre = Ptr1^.Degre)))
             then begin
                        If (Ptr1^.Categorie_Init = Nbre)
                           then OpFraction_Plus (Ptr1^.Val_Nbre, Ptr3^.Val_Nbre)
                           else OpFraction_Plus (Ptr1^.coefficient, Ptr3^.coefficient);
                        Ptr1^.suivant:=Ptr3^.suivant;
                        Dispose (Ptr2);
                        Supprime_Noeud(Ptr3);
                        Arbre_Syntaxique :=Ptr1;
                    end
             {Autres cas}
             else begin
                        Ptr2^.Suivant   := Ptr3^.suivant;
                        Ptr2^.POperand2 := Ptr1;
                        Ptr2^.POperand1 := Ptr3;
                        Ptr1^.suivant   := nil;
                        Ptr3^.suivant   := nil;
                        Arbre_Syntaxique   := Ptr2
                    end;
        Arbre_Syntaxique^.Categorie_Evol := Membre
end;
{ ------------------------- Fin Reduct_10 ---------------------------------- }


procedure Reduct_12;     { <Valeur> ::= <-> <(> <Membre> <)> }

var Ptr1,Ptr2,Ptr3,Ptr4,Ptr5 : PArbre; {Pointeurs temporaires}

begin
        Ptr1 := Arbre_Syntaxique;
        Ptr2 := Arbre_Syntaxique^.suivant;
        Ptr3 := Arbre_Syntaxique^.suivant^.suivant;
        If (Ptr2^.Categorie_Init = Nbre) or (Ptr2^.Categorie_Init = Inc)
           then begin
                        if (Ptr2^.Categorie_Init = Inc)
                        then Ptr2^.coefficient.numerateur
                                := (-1)* Ptr2^.coefficient.numerateur
                        else Ptr2^.Val_Nbre.numerateur
                                := (-1)* Ptr2^.Val_Nbre.numerateur;
                        Ptr2^.suivant := Ptr3^.suivant^.suivant;
                        Arbre_Syntaxique := Ptr2
                end
           else begin
                        new(Ptr4);
                        new(Ptr5);
                        Ptr5^.Categorie_Init := Nbre;
                        Ptr5^.Categorie_Evol := Nbre;
                        Ptr5^.suivant :=nil;
                        Ptr5^.Val_Nbre.numerateur :=-1;
                        Ptr5^.Val_Nbre.denominateur :=1;
                        Ptr4^.Categorie_Init := Fois;
                        Ptr4^.Categorie_Evol := Valeur;
                        Ptr4^.suivant:=Ptr3^.suivant^.suivant;
                        Ptr4^.POperand1:=Ptr5;
                        Ptr4^.POperand2 :=Ptr2;
                        Ptr2^.suivant:=nil;
                        Arbre_Syntaxique := Ptr4
                end;
        dispose (Ptr1);
        dispose (Ptr3^.suivant);
        dispose (Ptr3);
        Arbre_Syntaxique^.Categorie_Evol :=Valeur
End;
{ ------------------------- Fin Reduct_12 ---------------------------------- }


Procedure Reduct_13;     { <Valeur> ::=  <(> <Membre> <)> }

var Ptr1,Ptr2,Ptr3: PArbre;  {Pointeurs temporaires}

Begin
        Ptr1 := Arbre_Syntaxique;
        Ptr2 := Arbre_Syntaxique^.suivant;
```

```
    Ptr3 := Arbre_Syntaxique^.suivant^.suivant;
    Ptr2^.suivant := Ptr3^.suivant;
    dispose (Ptr1);
    dispose (Ptr3);
    Arbre_Syntaxique := Ptr2;
    Arbre_Syntaxique^.Categorie_Evol :=Valeur
ıd;
{ ------------------------- Fin Reduct_13 ---------------------------------- }


{****************** Début de Réductions ***************************}

Begin
  Case Recherche_Regle of
       0 : begin
                Erreur := true;
                Msg:= 'Il y a une erreur syntaxique dans l''équation !'
           end;
       1 : Reduct_1;          { <Valeur> ::= - <Inc> }
       2 : Arbre_Syntaxique^.Categorie_Evol := Valeur; { <Valeur> ::= <Inc> }
       3 : Reduct_3;          { <Valeur> ::= - <Nbre> }
       4 : Arbre_Syntaxique^.Categorie_Evol := Valeur; { <Valeur> ::= <Nbre> }
       5 : Reduct_5;          { <Facteur> ::= <Facteur> ^ <Valeur> }
       6 : Arbre_Syntaxique^.Categorie_Evol := Facteur; { <Facteur> ::= <Valeur> }
       7 : Reduct_7;          { <Terme> ::= <Terme> * <Facteur> }
       8 : Reduct_8;          { <Terme> ::= <Terme> / <Facteur> }
       9 : Arbre_Syntaxique^.Categorie_Evol := Terme; { <Terme> ::= <Facteur> }
       10 : Reduct_10;         { <Membre> ::= <Membre> <addop> <Terme> }
       11 : Arbre_Syntaxique^.Categorie_Evol := Membre; {<Membre>::=<Terme>}
       12 : Reduct_12;        { <Valeur> ::= - ( <Membre> ) }
       13 : Reduct_13;        { <Valeur> ::=   ( <Membre> ) }
  end;
End;
{ ------------------------- Fin Reductions ---------------------------------- }


Procedure Analyse_Membre ( Equ_dollar : string; var Erreur : boolean;
                           Num_Mbre : integer; var Msg : PChar;
                           var i : integer; var Arbre_Syntaxique : PArbre);

{ But : Transforme une partie de l'expression mathématique introduite par
        l'utilisateur, de sa représentation sous forme de caractères vers sa
        représentation sous forme d'arbre algébrique                          }

Type
    TTab_Dec = (A,D,E,P,R); {Valeurs possibles pour la table de décision}

Const
     Table_Decision : array [Membre .. Rien, Inc .. Rien] of TTab_Dec
                           = ((E,E,D,D,E,E,E,E,D,R),
                              (E,E,R,R,D,D,E,E,R,R),
                              (E,E,R,R,R,R,D,E,R,R),
                              (E,E,R,R,R,R,R,E,R,R),
                              (E,E,R,R,R,R,R,E,R,R),
                              (A,E,R,R,R,R,R,E,R,R),
                              (D,D,E,D,E,E,E,E,D,E,E),
                              (D,D,E,D,E,E,E,E,D,E,E),
                              (D,D,E,D,E,E,E,E,D,E,E),
                              (D,D,E,D,E,E,E,E,D,E,E),
                              (P,D,E,D,E,E,E,E,D,E,E),
                              (D,D,E,D,E,E,E,E,D,E,E),
                              (E,E,R,R,R,R,R,E,R,R),
                              (D,D,E,D,E,E,E,E,D,E,E));
      { Table de désision
        Les colonnes représentent les éléments de Arbre_Lexico
        Les lignes représentent les éléments de Arbre_Syntaxique }

ır

   Fin_Membre : boolean; { Indicateur de Fin d'analyse d'un membre }
   Arbre_Lexico :PArbre; { Pile composée de noeuds, constituée de symboles
                           terminaux auquels nous n'avons pas encore esssayé
                           d'appliquer une règle syntaxique }


ocedure Ajoute_Fois;
{ But : Cette procédure insère un élément <Fois> à Arbre_Lexico}

ır Ajout_Noeud :PArbre; {Element de type <Fois> que nous devons ajouter}
```

```
egin
    new(Ajout_Noeud);
    Ajout_Noeud^.POperand1:=nil;
    Ajout_Noeud^.POperand2:=nil;
    Ajout_Noeud^.Suivant:=Arbre_Lexico;
    Ajout_Noeud^.Categorie_Init:=Fois;
    Ajout_Noeud^.Categorie_Evol:=Fois;
    Arbre_Lexico:=Ajout_Noeud
nd;
{------------------------------ Fin Ajoute_Fois ------------------------------ }


rocedure Puissance_Inconnue (Categorie : TCategorie; var Msg : PChar);
{ But : Cette procedure initialise Msg en fonction de Categorie        }

Begin
    If (Categorie = Inc)
        then Msg := 'Nous sommes en train de résoudre un '
                  + 'problème du premier degré(ax+b =0).'
                  + ' Essaie de ne pas élever l''inconnue'
                  + ' à une puissance différente de -1,0 ou de 1.'

        else if (categorie = Nbre)
             then Msg := 'Cet exercice ne demande pas d''élever un nombre '
                  + 'à une puissance X. X représente l''inconnue dans '
                  + 'l''équation.'
             else Msg := 'Cet exercice ne demande pas d''élever une expression '
                  + 'à une puissance X. X représente l''inconnue dans '
                  + 'l''équation.';
nd;
{ ----------------- Fin Puissance_Inconnue ---------------------------------- }


{ ----------------- Début Analyse_Membre ------------------------------------ }
egin
     Fin_Membre :=false;
     New (Arbre_Lexico);
     New(Arbre_Syntaxique);
     Arbre_Syntaxique^.Categorie_Init := Rien;
     Arbre_Syntaxique^.Categorie_Evol := Rien;
     Arbre_Syntaxique^.Suivant:= nil;
     Arbre_Lexico^.Categorie_Evol := Rien;
     Arbre_Lexico^.Categorie_Init := Rien;
     Arbre_Lexico^.Suivant:= nil;
     While (Arbre_Syntaxique^.Categorie_Evol = Rien) and (not Erreur)
       and (not Fin_Membre) do
       AnalLex_et_Deplace_Objet(Symbole,Msg,Erreur,
                                Arbre_Syntaxique,Arbre_Lexico,Num_Mbre);
     If Erreur then exit;
     Repeat
       begin
         case Table_Decision [Arbre_Syntaxique^.Categorie_Evol,
                              Arbre_Lexico^.Categorie_Evol]of
              A : Ajoute_Fois;
              D : AnalLex_et_Deplace_Objet (Symbole,Msg,Erreur,
                                            Arbre_Syntaxique,Arbre_Lexico,
                                            Num_mbre);
              E : Erreur := true;
              P : begin Puissance_Inconnue
                        (Arbre_Syntaxique^.suivant^.Categorie_Init,Msg);
                         Erreur := true end;
              R : Reductions(Arbre_Syntaxique,Msg,Erreur);
          end;
       end;
     Until Erreur or (Arbre_Lexico^.Categorie_Evol=Rien);
     if Erreur
        then begin
               if Table_Decision [Arbre_Syntaxique^.Categorie_Evol,
                                  Arbre_Lexico^.Categorie_Evol] = E
               then
               Msg := 'Que signifie cette expression mathématique ?';
               exit;
             end;
     while (Arbre_Syntaxique^.Categorie_Evol<>Membre)do
      begin
            Reductions(Arbre_Syntaxique,Msg,Erreur);
            if erreur then exit;
      end;
      If (Arbre_Syntaxique^.suivant^.Categorie_Evol<>Rien)
```

```
          then
              begin
                   if Arbre_Syntaxique^.suivant^.Categorie_Evol=ParG
                      then Msg:='As-tu mis tes parenthèses correctement ?'
                      else Msg:='Que signifie cette expression mathématique ?';
                   Erreur:=true;
                   exit;
              end
End;
  ------------------------ Fin Analyse_Membre ------------------------------- }


rocedure Parcours_Membre(Mbre: PArbre; inconnue : char; var Cpt_Inc : integer);
 But : Cette procédure parcourt Mbre et compte le nombre d'occurences
       d'inconnue}

gin
    If (Mbre^.Categorie_Init = Inc) or (Mbre^.Categorie_Init = Nbre)
       then begin
              if (Mbre^.Categorie_Init = Inc)
               then Cpt_Inc := Cpt_Inc +1
            end
       else begin
                 Parcours_Membre (mbre^.Poperand1,inconnue,cpt_inc);
                 Parcours_Membre (mbre^.Poperand2,inconnue,cpt_inc)
            end;
end;
{ ------------------------- Fin Parcours_Membre ---------------------------- }


Procedure Analyseur_Syntaxique (var Msg : PChar; var Erreur : boolean;
                                var arbre_equation : Tab_Membres;
                                    inconnue : char; Equation : string);

{ But : Cette procédure représente l'implémentation de notre analyseur
       syntaxique. Elle détermine si une expression introduite par
       l'utilisateur est syntaxiquement correcte ou pas}

var Cpt_inc    : integer; {Compteur du nombre d'occurences d'un terme contenant
                           l'inconnue dans Equ_dollar}
    Num_mbre   : integer; { Indicateur permettant de connaître le membre de
                            l'équation qui est en cours d'analyse          }

Begin
  i := 1;
  Equ_dollar := Equation+'$';
  Num_Mbre:=1;
  While not erreur and (Num_Mbre < 3) do
    begin
      Symbole := Char_Suivant(Equ_dollar,i);
      Analyse_Membre(Equ_dollar,erreur,Num_Mbre,Msg,i,
                     arbre_equation[num_mbre]);
      Num_Mbre:=Num_Mbre+1
    end;
if not erreur
    then begin
         Cpt_Inc := 0;
         Parcours_Membre(Arbre_Equation[1],inconnue,Cpt_Inc);
         Parcours_Membre(Arbre_Equation[2],inconnue,Cpt_Inc);
         if (Cpt_Inc < 1)
            then Msg :='Ce n''est pas une équation. Il n''y a pas d''inconnue.';
         end;
nd;
{ ------------------------- Fin Analyseur_Syntaxique ----------------------- }

 GIN

END.
```

```
{*************************************************************************}
{*                              UNIT CANON                                 *}
{*                                                                         *}
{* Cette unité regroupe toutes les procédures relatives à la résolution de *}
{* l'équation introduite par le professeur (mise sous forme canonique et   *}
{* solution de l'équation) :                                               *}
{* - Construit_Pile                                                        *}
{* - Effectue                                                              *}
{* - Reduit_Equation                                                       *}
{* - Recherche_Equation_Canonique                                          *}
{* - Solution_Equation                                                     *}
{*************************************************************************}

unit Canon;

{ ************************************************************************** }
{ ********************** INTERFACE ***************************************** }
  ************************************************************************** }

INTERFACE

uses Fraction,Syntaxe,Sysutils;

{ ************************** Déclaration de la constante ******************}
Const Maxpile = 50;                 {Taille maximum d'une pile}


{ ************************** Déclaration des types ************************}
Type
    Tnature = (nb,inco,operateur);  {Le type des éléments contenue par une pile}
    Telement = record
                 case  elem : Tnature of
                       nb : (fract : Tfraction);
                       inco  : (coef : TFraction; Degre : Integer);
                       operateur : (oper : TCategorie);
                 end;
    { nb représente un nombre
      inco représente un terme contenant l'inconnue
      operateur représente + - * / ^                }

    Telt = record
            prec : boolean;
            case  elem : Tnature of
                  nb : (fr : Tfraction);
                  inco  : (coef : TFraction; Deg : Integer);
                  Plus,Moins,Fois,Divis,Puiss : (op : TCategorie);
            end;

    Tequ_canon = array [1..2] of Telt;
    Tpile = array [1..maxpile] of Telement;
    Tpilev = array [1..maxpile] of Telt;
    Tpilinc = array [0..maxpile] of Telt;
    TConst = 1..maxpile;


  ************************** Déclaration des variables globales ************}
Var lpile      : Tconst; {Hauteur réelle de la pile}
    p          : Tpile;  {Pile contenant un Arbre lu de manière postfixée}
    pilev      : Tpilev; {Pile d'évaluation}
    tampon     : Telt;   {Pile tampon}


  ************************** Déclaration des entêtes **********************}
Procedure Construit_Pile
(var p : Tpile; var lpile : Tconst;Arbre : Parbre);

Procedure Effectue (un,deux : Telt; var temp :Telt;var msg : PChar;
                    var erreur : boolean; var tampon : Telt);

Procedure Reduit_Equation
        (var msg : PChar;var erreur : boolean;p:tpile;lpile:integer;
         var pilev : Tpilev; var tampon : Telt);

Procedure Recherche_Equation_canonique
(var equ_canonique : Tequ_canon; var msg : PChar;var erreur : boolean;
  ampon : telt;pilev :Tpilev );

Procedure solution_equation (var solution : Tfraction; var msg : PChar;
                             equ_canonique : Tequ_canon;var erreur : boolean);
```

```
{ ************************************************************************** }
{ ********************* IMPLEMENTATION ********************************** }
{ ************************************************************************** }

MPLEMENTATION


rocedure Construit_Pile (var p : Tpile; var lpile : Tconst;
                         Arbre : Parbre);
{Cette procédure transforme une expression mise sous forme infixée à la forme
 postfixée en construisant une pile}

egin
    If (Arbre^.Categorie_Init = Inc) or (Arbre^.Categorie_Init = Nbre)
       then
         {Si l'élement est un terme contenant l'inconnue}
         if (Arbre^.Categorie_Init = Inc)
            then begin
                   p[lpile].elem :=inco;
                   p[lpile].coef :=Arbre^.coefficient;
                   p[lpile].degre := Arbre^.degre;
                   lpile := lpile +1
                 end
            else begin
            {Si l'élement est un nombre}
                   p[lpile].elem :=nb;
                   p[lpile].fract:=Arbre^.Val_nbre;
                   lpile := lpile +1
                 end
       else begin
              {Si l'élement est un opérateur}
              Construit_Pile(p,lpile,Arbre^.Poperand1);
              Construit_Pile(p,lpile,Arbre^.Poperand2);
              if (Arbre^.Categorie_Init in [Plus..Puiss])
                 then begin
                        p[lpile].elem := operateur;
                        p[lpile].oper := Arbre^.Categorie_Init;
                        lpile := lpile +1
                      end;
            end;
nd;
  ----------------------- Fin Construit_Pile ---------------------------------- }



rocedure Effectue (un,deux : Telt; var temp :Telt;var msg : PChar;
                   var erreur : boolean; var tampon : Telt);

{ But : Cette procédure effectue des opérations sur des nombres et des termes
        contenant l'inconnue}

begin
  { < Op > ::= <Plus> | <Moins> | <Fois> | <Divis> | <Puiss> }
  {<Nbre> < Op > < Nbre > }
  If (temp.elem = nb) and (un.elem = nb)
     then begin
          case deux.op of
    Plus,Moins : begin
                   if deux.op = Moins then un.fr.numerateur :=(-1)*un.fr.numerateur;
                   OpFraction_Plus (temp.fr,un.fr);
                 end;
         Fois : OpFraction_Fois (temp.fr,un.fr);
        Divis : OpFraction_Divis(temp.fr,un.fr,Msg,Erreur);
        Puiss : OpFraction_Puiss(temp.fr,un.fr,Msg,erreur);
         end
         end
    else begin
    { <Inc> < Op > < Nbre > }
         if (temp.elem = inco) and (un.elem = nb)
         then begin
              case deux.op of
        Plus,Moins : begin
                     if deux.op = Moins then un.fr.numerateur :=(-1)*un.fr.numerateur;
                     tampon:=temp;
                     temp.elem:=nb;
                     temp.fr:=un.fr
                     end;
             Fois : OpFraction_Fois (temp.coef,un.fr);
            Divis : OpFraction_Divis(temp.coef,un.fr,Msg,Erreur);
```

```
      Puiss : begin
              if (- un.fr.numerateur= un.fr.denominateur)
              then temp.deg := (-1)*temp.deg
              else begin
                   if un.fr.numerateur = 0
                   then begin
                        temp.elem:=nb;
                        temp.fr.numerateur:=1;
                        temp.fr.denominateur:=1
                        end
                   else if not (un.fr.numerateur = un.fr.denominateur)
                        then begin
                             Msg := 'Nous sommmes en train de '
                             +'résoudre un problème du premier degré'
                             +' (ax+b=0). Essaie de ne pas élever '
                             +'l''inconnue à une puissance différente '
                             +'de -1,0 ou de 1.';
                             Erreur:= true;
                             end;
                   end;
              end;
       end;
       end
  else begin
 { <Nbre> < Op > < Inc > }
  if (temp.elem = nb) and (un.elem = inco)
     then begin
          case deux.op of
   Plus, Moins : begin
                 if deux.op = Moins
                 then un.coef.numerateur :=(-1)*un.coef.numerateur;
                 tampon:=un;
                 end;
          Fois : begin
                 OpFraction_Fois (un.coef,temp.fr);
                 temp:=un
                 end;
         Divis : begin
                 OpFraction_Divis(temp.fr,un.coef,Msg,Erreur);
                 if erreur then exit;
                 un.coef:=temp.fr;
                 temp:=un
                 end;
         puiss : begin
                 Msg :='Cet exercice ne demande pas d''élever'
                          +' un nombre à une puissance X.'
                          +' X  représente l''inconnue dans '
                          +'l''équation.';
                 Erreur := true;
                 end;
          end;
          end
     else begin
     { <Inc> < Op > < Inc > }
     if (temp.elem = inco) and (un.elem = inco)
        then begin
             case deux.op of
       Plus,Moins : begin
                    if deux.op = moins
                    then un.coef.numerateur :=(-1)*un.coef.numerateur;
                    If temp.deg = un.deg
                    then Opfraction_plus(Temp.coef,un.coef)
                    else begin
                    Msg := 'En résolvant cette équation, tu vas '
                              +'aboutir à une expression de la '
                              +'forme ax+a/x, où x représente ton '
                              +'inconnue et ''a''son coefficient. '
                              +'Tu vas remettre l''expression au '
                              +'même dénominateur et tu ne vas plus '
                              +'une équation du premier degre.';
                    Erreur := true
                         end;
                    end;
               Fois : begin
                    Opfraction_fois (Temp.coef,un.coef);
                    temp.deg := temp.deg + un.deg;
                    If temp.deg = 0
                       then begin
                            tampon.fr:=temp.coef;
                            temp:=tampon
```

```
                                end
                            else begin
                              if (temp.deg <>1) and (temp.deg <>-1)
                              then begin
                                   Msg := 'Nous sommmes en train de résoudre'
                                            +' un problème du premier degré '
                                            +'(ax+b=0). Essaie de ne pas '
                                            +'multiplier l''inconnue par '
                                            +'elle-même.';
                                   Erreur:= true;
                                   end;
                              end;
                        end;
                  Divis : begin
                          Opfraction_divis(Temp.coef,un.coef,msg,erreur);
                          If erreur then exit;
                          temp.deg := temp.deg - un.deg;
                          if temp.deg = 0
                             then begin
                                  tampon.fr:=temp.coef;
                                  temp:=tampon
                                  end
                             else begin
                                  if (temp.deg <>1) and (temp.deg <>-1)
                                  then begin
                                  Msg := 'Nous sommmes en train de résoudre'
                                            +' un problème du premier degré '
                                            +'(ax+b=0). Essaie de ne pas diviser'
                                            +' l''inconnue par elle_même.';
                                  Erreur:= true;
                                       end;
                                  end;
                          end;
                  Puiss : begin
                          Msg := 'Nous sommmes en train de résoudre un '
                                   +'problème du premier degré (ax+b=0). '
                                   +'Essaie de ne pas élever l''inconnue'
                                   +' à une puissance différente de -1,0 '
                                   +'ou de 1.';
                          Erreur:= true;
                          end;
                   end;
                   end
                end;
             end;
          end;
nd;
 ------------------------ Fin Effectue ------------------------------------ }


rocedure Reduit_Equation
        (var msg : PChar;var erreur : boolean;p:tpile;lpile:integer;
         var pilev : Tpilev; var tampon : Telt);

9ut : Cette procédure construit deux piles tampon et pilev}

Var
    avant  : boolean;     {Valeur booléenne indiquant s'il faut remettre
                           l'élément sur la pile ou pas }
    encore : boolean;     {Valeur booléenne indiquant s'il faut extraire
                           un nouvel élément de la pile p ou pas          }
    Sommet : 0..maxpile;  {Sommet de la pile pilev}
    cas    : Tnature;     {Type de l'élément}
    temp   : Telt;        {Tampon}
    pilinc : Tpilinc;     {Pile composée uniquement de termes contenant l'inconnue}
    un,deux : Telt;       {éléments d'une pile}
    i,j     : integer;    {Indices de pile}


Procedure push (entree : Telt);
' But : Cette procédure ajoute un élément "entree" à la pile pilev}

:gin
     if sommet = maxpile
        then begin
               msg := 'Crois-tu vraiment que le problème a autant de'
                        +' données ?';
               erreur := true;
               halt;
            end
```

```
        else begin
               sommet := sommet +1;
               pilev[sommet] := entree
             end
nd;
  ---------------------- Fin Push ------------------------------------------ }

Procedure pop (var sortie : Telt);
  But : Cette procédure retire un élément "sortie" de la pile pilev }
egin
     if sommet = 0
        then begin
               msg := 'Pile vide';
               erreur := true;
               exit;
             end
        else begin
               sortie :=pilev[sommet];
               sommet := sommet -1;
             end
nd;
  -------------------- Fin Pop ------------------------------------------- }

{ -------------------- Début Reduit_Equation ---------------------------------- }
egin
  avant := false;
  sommet:=0;
  erreur:=false;
  msg:='';
  temp.deg := 0;
  For j := lpile-1 downto 1 do pilinc[j].elem :=nb;
  j:=1;
  For i := lpile-1 downto 1 do
   begin
     encore :=false;
     cas := p[i].elem;
     repeat
       case cas of
            {L'élément de la pile p est un opérateur}
            operateur : begin
                          un.prec := avant;
                          un.elem := operateur;
                          un.op := p[i].oper;
                          push(un);
                          avant :=false
                        end;
            {L'élément de la pile p est un nombre ou un terme contenant l'inconnue}
            nb, inco : begin
                         if not encore
                            then if cas = nb
                                 then begin
                                   temp.elem :=nb;
                                   temp.fr:= p[i].fract;
                                      end
                                 else begin
                                   temp.elem:= inco;
                                   temp.coef:=p[i].coef;
                                   temp.deg := p[i].degre;
                                      end;
                       if not avant
                          then begin
                               un.prec := avant;
                               if temp.elem = nb
                                  then begin
                                       un.elem :=nb;
                                       un.fr:=temp.fr;
                                       end
                                  else begin
                                       un.elem :=inco;
                                       un.coef := temp.coef;
                                       un.deg :=temp.deg
                                       end;
                               push(un);
                               encore :=false;
                               avant :=true
                               end
                          else begin
                               pop(un);
                               pop(deux);
                               tampon.elem :=nb;
```

```
                    if tampon.coef.numerateur <> abs (tampon.coef.numerateur)
                    then tampon.coef.numerateur := (-1)* tampon.coef.numerateur;
                    equ_canonique[1]:=tampon;
                    equ_canonique[2].fr.numerateur:=0;
                    equ_canonique[2].fr.denominateur:=1;
                    end
                 else begin
                    Msg := 'En résolvant cette équation, tu vas '
                              +'aboutir à une expression de la '
                              +'forme ax+a/x, où x représente ton '
                              +'inconnue et ''a''son coefficient. '
                              +'Tu vas remettre l''expression au '
                              +'même dénominateur et tu n''auras '
                              +'une équation du premier degre.';
                    Erreur := true
                    end
              end
           else begin
              if pilev[1].coef.numerateur <> abs (pilev[1].coef.numerateur)
              then pilev[1].coef.numerateur := (-1)*pilev[1].coef.numerateur;
              equ_canonique[1]:=pilev[1];
              equ_canonique[2].fr.numerateur:=0;
              equ_canonique[2].fr.denominateur:=1;
              end
        end
nd;
 ------------- Recherche_Equation_Canonique ------------------------------- }


rocedure solution_equation (var solution : Tfraction; var msg : PChar;
                            equ_canonique : Tequ_canon;var erreur : boolean);

{But : Cette procédure calcule la solution de l'équation canonique.}

ar degre : integer;    {Variable "bidon" nécessaire si nous souhaitons utiliser
                        la procédure Inverse_Fraction pour un nombre}
   sol_str : PChar;   {Chaîne de caractères}

egin
    If (equ_canonique[1].coef.numerateur = 0) and
       (equ_canonique[2].fr.numerateur = 0)
       then msg := 'Dans R, cette équation admet tout réel comme '
                     +'solution : S = R. L''équation est dite "indéterminée".'
       else begin
       If (equ_canonique[1].coef.numerateur = 0) and
          (equ_canonique[2].fr.numerateur <> 0)
       then msg := 'Dans R, l''équation n''admet aucun réel comme '
                     +'solution : S = {}. L''équation est dite "impossible".'
       else begin
            solution:=equ_canonique[2].fr;
            solution.numerateur:=(-1)*solution.numerateur;
            Inverse_Fraction (equ_canonique[1].coef,nbre,degre,Erreur,
            msg);
            if erreur then exit;
            Opfraction_fois(solution,equ_canonique[1].coef);
            str(round(solution.numerateur),sol_str);
            msg :='La solution de l''équation : S = {' + sol_str;
            if (solution.denominateur <> 1)
               then begin
                  str(round(solution.denominateur),sol_str);
                  msg := msg + '/'+sol_str+'}';
                    end
               else msg:=message+'}';
            end;
            end;
End;
 ----------------- Fin Solution_Equation ------------------------------------ }

egin
end.
```

```
{*****************************************************************************}
{*                                UNIT SEMANTIC                              *}
{*                                                                           *}
{* Cette unité regroupe toutes les fonctions et les procédures relatives à   *}
{* l'implémentation de l'analyseur sémantique  :                             *}
{* - Neutre                                                                  *}
{* - Opposé                                                                  *}
{* - Inverse                                                                 *}
{* - Transforme_Moins_en_Plus                                                *}
{* - Transforme_Divis_en_Fois                                                *}
{* - Meme_denominateur                                                       *}
{* - Un_seul_membre                                                          *}
{* - Ramene_Operateur_a_gauche                                               *}
{* - Compare                                                                 *}
{* - Associativite                                                           *}
{* - Comparaison_Semantique                                                  *}
{* - Commutativite                                                           *}
{* - Transforme_en_Plus                                                      *}
{* - Compare_Equation                                                        *}
{* - Analyseur_Semantique                                                    *}
{*****************************************************************************}

UNIT SEMANTIC;

{ ************************************************************************** }
{ ********************** INTERFACE ***************************************** }
{ ************************************************************************** }

INTERFACE

uses FRACTION,SYNTAXE;

{ ************************** Déclaration des types ************************** }
type PListe = ^Tliste;
     TListe = record
                    elem : PArbre;
                    suivant : PListe
              end;

{ ************************** Déclarations de entêtes ************************ }
Function Neutre (Arbre : PArbre):PArbre;
Function Oppose (Arbre : PArbre): PArbre;
Procedure Inverse (var Arbre : PArbre; var Msg : PChar;
                   var Erreur : boolean);
Procedure Transforme_Moins_en_Plus (var Arbre : PArbre);
Procedure Transforme_Divis_en_Fois (var Arbre : PArbre;
          var Msg : PChar; var Erreur : boolean);
Procedure Meme_Denominateur (var Arbre_Equation : Tab_Membres);
Function Un_seul_Membre (Arbre_Equation : Tab_Membres) : PArbre;
Procedure Ramene_Operateur_A_gauche (var Arbre : PArbre;
          var Msg : PChar;var Erreur : boolean);
Procedure FractREAL (elem1,elem2 : PArbre; var fract1,fract2 : real);
Function compare (elem1,elem2 : PArbre; var egal : boolean) : boolean;
Procedure Associativite (var Arbre:PArbre);
Function Comparaison_Semantique (Arbre1,Arbre2 : PArbre) : Boolean;
Procedure Commutativite (var Arbre : PArbre; var Msg : PChar;
                         var Erreur : boolean);
Function Transforme_en_Plus (arbre : Parbre; var Msg : PChar;
                             var erreur : boolean) : Parbre;

{ ************************************************************************** }
{ ********************** IMPLEMENTATION ********************************** }
{ ************************************************************************** }

IMPLEMENTATION


Function Neutre (Arbre : PArbre):PArbre;
{ But : Cette procédure applique la propriété du neutre pour l'addition et pour
        la multiplication à Arbre}

var Sous_arbre : PArbre;  { Variable "bidon" qui représnte le sous-arbre droit
                            d'Arbre }

begin
    While {Propriété pour l'addition et la soustraction}
          (((Arbre^.Categorie_Init in [Plus .. Moins]) and
            (Arbre^.POperand2^.Categorie_Init = Nbre)  and
            (Arbre^.POperand2^.Val_Nbre.numerateur = 0))
```

```
      or
         {Propriété pour la multiplication, la division et la puissance}
         ((Arbre^.Categorie_Init in [Fois .. Puiss]) and
          (Arbre^.POperand2^.Categorie_Init = Nbre)  and
          (  Arbre^.POperand2^.Val_Nbre.numerateur
           = Arbre^.POperand2^.Val_Nbre.denominateur)))do
      begin
           Sous_Arbre := Arbre^.POperand1;
           Supprime_Noeud (Arbre^.POperand2);
           dispose(Arbre);
           Arbre:=Sous_arbre
      end;

     While {Propriété pour l'addition et la soustraction}
        (((Arbre^.Categorie_Init in [Plus .. Moins]) and
          (Arbre^.POperand1^.Categorie_Init = Nbre)  and
          (Arbre^.POperand1^.Val_Nbre.numerateur = 0))
       or
         {Propriété pour la multiplication, la division et la puissance}
         ((Arbre^.Categorie_Init in [Fois .. Puiss]) and
          (Arbre^.POperand1^.Categorie_Init = Nbre)  and
          (  Arbre^.POperand1^.Val_Nbre.numerateur
           = Arbre^.POperand1^.Val_Nbre.denominateur)))do
      begin
           Sous_Arbre := Arbre^.POperand2;
           Supprime_Noeud (Arbre^.POperand1);
           dispose(Arbre);
           Arbre:=Sous_arbre
      end;
    Neutre:=Arbre
nd;
{ ------------------------ Fin Neutre ------------------------------------ }


unction Oppose (Arbre : PArbre): PArbre;

ar Sous_Arbre : PArbre; {Variable 'bidon' contenant un sous-arbre d'Arbre}

egin
     Case Arbre^.Categorie_Init of

        {L'opposé d'un nombre (ou d'un terme contenant l'inconnue) s'obtient
         en multipliant celui-ci (ou le coefficient de l'inconnue) par -1.}
        Nbre : Arbre^.Val_Nbre.Numerateur := (-1)*Arbre^.Val_Nbre.Numerateur;

        Inc  : Arbre^.coefficient.Numerateur := (-1)*Arbre^.coefficient.Numerateur;

        {L'opposé d'une somme (de "+" ou de "-") s'obtient en faisant la somme
         des opposés.}
        Plus, Moins : begin
                         Arbre^.POperand1:=oppose(Arbre^.POperand1);
                         Arbre^.POperand2:=oppose(Arbre^.POperand2)
                      end;

        {L'opposé d'un produit (de "*" ou de "/") est le produit de l'opposé
         d'un facteur par l'autre facteur ou les autres.}
        Fois, Divis : begin
                      if (Arbre^.POperand1^.Categorie_Init = Nbre)
                         then begin
                           if(- Arbre^.POperand1^.Val_Nbre.numerateur
                              = Arbre^.POperand1^.Val_Nbre.denominateur)
                                 then begin
                                        Sous_arbre := Arbre^.POperand2;
                                        Supprime_Noeud (Arbre^.POperand1);
                                        dispose(Arbre);
                                        Arbre:=Sous_arbre
                                      end
                                 else Arbre^.POperand1^.Val_Nbre.numerateur :=
                                    (-1)*Arbre^.POperand1^.Val_Nbre.numerateur;
                           end
                      else if (Arbre^.POperand1^.Categorie_Init = Nbre)
                            then begin
                                 Arbre^.POperand2^.Val_Nbre.numerateur :=
                                 (-1)*Arbre^.POperand2^.Val_Nbre.numerateur;
                                 if(- Arbre^.POperand1^.Val_Nbre.numerateur
                                  = Arbre^.POperand1^.Val_Nbre.denominateur)
                                 then Arbre:=Neutre(Arbre);
                                 end
                            else Arbre^.POperand1:=oppose(Arbre^.POperand1);
```

```
                    end;

            {Multiplication du SAD par -1}
            Puiss : begin
                        new(sous_arbre);
                        sous_arbre^.POperand1:=Arbre;
                        sous_arbre^.Categorie_Init:=Fois;
                        sous_arbre^.suivant:=nil;
                        new(sous_arbre^.POperand2);
                        sous_arbre^.POperand2^.Val_Nbre.numerateur:=-1;
                        sous_arbre^.POperand2^.Val_Nbre.denominateur:=1;
                        sous_arbre^.POperand2^.Categorie_Init:=Nbre;
                        sous_arbre^.POperand2^.suivant:=nil;
                        Arbre:=sous_arbre;
                    end;
    End;
    oppose:=Arbre;
nd;
 ------------------------- Fin Oppose ---------------------------------- }



rocedure Inverse (var Arbre : PArbre; var Msg : PChar;
                  var Erreur : boolean);

ar Sous_Arbre  : PArbre;  {Variable 'bidon' contenant un sous-arbre d'Arbre}
   Simul_degre : integer; {Variable 'bidon' nécessaire si nous souhaitons
                           utiliser la procédure Inverse_Fraction pour un <NBre>}

egin
    Case Arbre^.Categorie_Init of

       {L'inverse d'un nombre n est égal à 1/n.}
        Nbre : Inverse_Fraction(Arbre^.Val_nbre,Nbre,Simul_Degre,Erreur,Msg);

       {L'inverse d'un terme contenant l'inconnue s'obtient en inversant son
        coefficient et en multipliant la puissance de l'inconnue par -1.}
        Inc : Inverse_Fraction(Arbre^.coefficient,Inc,Arbre^.Degre,Erreur,Msg);

       {L'inverse d'une somme (de "+" et de "-") s'obtient en élevant la somme
        à la puissance -1.}
        Plus, Moins : begin
                        new(sous_arbre);
                        sous_arbre^.POperand1:=Arbre;
                        sous_arbre^.Categorie_Init:=Puiss;
                        sous_arbre^.suivant:=nil;
                        new(sous_arbre^.POperand2);
                        sous_arbre^.POperand2^.Val_Nbre.numerateur:=-1;
                        sous_arbre^.POperand2^.Val_Nbre.denominateur:=1;
                        sous_arbre^.POperand2^.Categorie_Init:=Nbre;
                        sous_arbre^.POperand2^.suivant:=nil;
                        Arbre:=sous_arbre;
                      end;

       {L'inverse d'un produit s'obtient en faisant le produit des inverses.}
              Fois : begin
                        Inverse(Arbre^.POperand1,Msg,Erreur);
                        Inverse(Arbre^.POperand2,Msg,Erreur)
                      end;

       {L'inverse d'un quotient s'obtient en faisant le quotient des inverses.}
             Divis : begin
                        sous_arbre:=Arbre^.POperand2;
                        Arbre^.POperand2:=Arbre^.POperand1;
                        Arbre^.POperand1:=sous_arbre
                      end;

       {Calcul de l'opposé du SAD}
             Puiss : begin
                        if ((Arbre^.POperand2^.Categorie_Init = Nbre)
                         and(- Arbre^.POperand2^.Val_Nbre.numerateur
                             = Arbre^.POperand2^.Val_Nbre.denominateur))
                            then begin
                                  Arbre^.POperand2^.Val_Nbre.numerateur :=
                                 (-1)*Arbre^.POperand2^.Val_Nbre.numerateur;
                                  Arbre:=Neutre (Arbre);
                                 end
                           else Arbre^.POperand2:=oppose(Arbre^.POperand2);
                      end
    End;
```

```
nd;
 ------------------------------ Fin Inverse ------------------------------- }


rocedure Transforme_Moins_en_Plus (var Arbre : PArbre);
{But : Cette procédure supprime de l'arbre Arbre l'opérateur non
       commutatif '-'}

ar Sous_Arbre : PArbre; {Variable 'bidon' contenant un sous-arbre d'Arbre}

Begin
      while (Arbre^.Poperand2^.Categorie_Init = Plus) do
       begin
            Arbre^.Poperand2^.POperand2 :=Oppose(Arbre^.POperand2^.POperand2);
            sous_arbre:=Arbre^.POperand2;
            Arbre^.POperand2:=Arbre^.POperand2^.POperand1;
            sous_arbre^.POperand1:=Arbre^.POperand1;
            Arbre^.POperand1:=sous_arbre;
       end;
      Arbre^.POperand2:=oppose(Arbre^.POperand2);
      Arbre^.Categorie_Init:=Plus
nd;
{ -------------------------- Transforme_Moins_en_Fois ---------------------- }


Procedure Transforme_Divis_en_Fois (var Arbre : PArbre; var Msg : PChar;
                                    var Erreur : boolean    );
But : Cette procédure supprime de l'arbre Arbre l'opérateur non
       commutatif '/'}

Var Sous_Arbre : PArbre; {Variable 'bidon' contenant un sous-arbre d'Arbre}

egin
     while (Arbre^.POperand2^.Categorie_Init = Fois) do
       begin
            Inverse(Arbre^.POperand2^.POperand2,Msg,Erreur);
            sous_arbre:=Arbre^.POperand2;
            Arbre^.POperand2:=Arbre^.POperand2^.POperand1;
            sous_arbre^.POperand1:=Arbre^.POperand1;
            Arbre^.POperand1:=sous_arbre;
       end;
     inverse(Arbre^.POperand2,Msg,Erreur);
     Arbre^.Categorie_Init:=Fois;
nd;
 ---------------------- Transforme_Divis_en_Fois ------------------------- }


rocedure Meme_Denominateur (var Arbre_Equation : Tab_Membres);
 But : Cette procédure multiplie les éléments du membre de droite
       Arbre_Equation[2] par les éléments de Arbre_Equation[1]
       qui sont au mis au dénonminateur}

ar Arbre : PArbre; {Variable utilisée pour créer un élément <Fois>}

Begin
     new (Arbre);
     Arbre^.Categorie_Init := Fois;
     Arbre^.Categorie_Evol := Membre;
     Arbre^.suivant := nil;
     Arbre^.POperand1 := Arbre_Equation[2];
     Arbre^.POperand2 := Arbre_Equation[1]^.POperand2;
     Arbre_Equation[2] := Arbre;
     Arbre:= Arbre_Equation[1]^.POperand1;
     Arbre_Equation[1]^.POperand1 := nil;
     Arbre_Equation[1] := Arbre;
nd;
{ ---------------------- Fin Meme_denominateur ---------------------------- }


Function Un_seul_Membre (Arbre_Equation : Tab_Membres) : PArbre;
' But : Cette procédure ramène les éléments du membre droite dans le membre
       de gauche }

Var Arbre : PArbre; {Variable utilisée pour créer un élément <Moins>}

 gin
```

```
     If (Arbre_Equation[1]^.Categorie_init = Divis)
        then Meme_Denominateur(Arbre_equation);
     new (Arbre);
     Arbre^.Categorie_Init := Moins;
     Arbre^.Categorie_Evol := Membre;
     Arbre^.suivant := nil;
     Arbre^.POperand1 := Arbre_Equation[1];
     Arbre^.POperand2 := Arbre_Equation[2];
     Transforme_Moins_en_Plus (Arbre);
     Un_seul_Membre := Arbre;
end;
{ -------------------- Fin Un_Seul_Membre ---------------------------------- }


Procedure Ramene_Operateur_A_gauche (var Arbre : PArbre;
                                     var Msg : PChar;
                                     var Erreur : boolean);

{But : Cette procédure normalise Arbre en suppimant les opérateurs non
commutatifs '-' et '/' et en ramenant à gauche les opérateurs commutatifs.}

var Operateur  : TCategorie;
    Sous_Arbre : PArbre;
    Temp_Arbre : PArbre;

begin
     Operateur := Arbre^.Categorie_Init;
     Case Arbre^.Categorie_Init of
               Moins : Transforme_Moins_en_Plus (Arbre);
               Divis : Transforme_Divis_en_Fois (Arbre,Msg,Erreur);
          Plus, Fois : begin
                       If (Operateur = Arbre^.POperand2^.Categorie_Init)
                          then begin
                                  sous_arbre := Arbre^.POperand2;
                                  While (Operateur
                                        = Sous_Arbre^.POperand1^.Categorie_Init)
                                        do Sous_Arbre := Sous_Arbre^.POperand1;
                                  Temp_Arbre := Sous_Arbre^.POperand1;
                                  Sous_Arbre^.POperand1 := Arbre^.POperand1;
                                  Arbre^.POperand1 := Arbre^.POperand2;
                                  Arbre^.POperand2 := Temp_Arbre;
                               end
                       end
     End;
end;
  ------------------------ Fin Ramène_Operateur_a_gauche ---------------------- }


Procedure FractREAL (elem1,elem2 : PARbre; var fract1,fract2 : real);
 But : Cette procédure assigne à deux variables de type real, les valeurs
       des deux éléments mis sous forme fractionnelle                    }

begin
  if (elem1^.categorie_init = Nbre)
     then begin
          fract1:=((elem1^.Val_nbre.numerateur)
                    /(elem1^.Val_nbre.denominateur));
          fract2:=((elem2^.Val_nbre.numerateur)
                  /(elem2^.Val_nbre.denominateur))
          end
     else begin
          If (elem1^.degre = elem2^.degre)
             then begin
                             fract1:=((elem1^.Cofacteur.numerateur)
                                     /(elem1^.Cofacteur.denominateur));
                             fract2:=((elem2^.Cofacteur.numerateur)
                                     /(elem2^.Cofacteur.denominateur))
                  end
              else begin
                                fract1:= elem1^.degre;
                                fract2:= elem2^.degre
                           end;
           end;
end;
  ----------------- Fin FractReal ---------------------------------------- }


Function compare (elem1,elem2 : PArbre; var egal : boolean) : boolean;
 But : Cette procédure compare les valeurs de deux nombres réelles }
```

```
ar fract1,fract2 : real;

Begin
     egal := false;
     If (elem1^.Categorie_init = Elem2^.categorie_init)
        then begin
               FRACTREAL (elem1,elem2,fract1,fract2);
               if (fract1 <= fract2)
                 then begin
                      compare := true;
                      if fract1 = Fract2 then egal := true
                      end
                 else compare := false;
            end
        else compare := false;

nd;
 ------------------------------ Fin Compare ---------------------------------}



rocedure Associativite (var Arbre:PArbre);

type PListe = ^Tliste;
     TListe = record
                    elem : PArbre;
                    suivant : PListe
              end;

ar Liste : PListe;


unction Cree_Liste (Operateur : TCategorie; Arbre : PArbre): Pliste;

var Cellule : PListe;

egin
     new(Cellule);
     If (operateur <> Arbre^.Categorie_Init)
        then begin
                  Cellule^.elem := Arbre;
                  Cellule^.suivant := nil
             end
        else begin
                  Cellule^.elem := Arbre^.POperand2;
                  Cellule^.suivant := Cree_Liste(operateur,Arbre^.POperand1)
             end;
     Cree_Liste := Cellule;
End;

 ---------------------- Fin Cree_Liste ---------------------------------- }


unction Alterne_Cellule (elem1,elem2 : PArbre) : boolean;

Var temp_arbre : PArbre;
    egal       : boolean;

gin
     alterne_cellule := false;
     if ((elem1 <> nil) and (elem2 <>nil))
       then begin
              if ((elem1^.categorie_init in [Inc..Nbre])
               or (elem2^.categorie_init in [Inc..Nbre]))
                 then begin
                        if ((elem1^.categorie_init > elem2^.categorie_init)
                         or compare (elem1,elem2,egal))
                          then begin
                                    new(temp_arbre);
                                    temp_arbre^:=elem2^;
                                    elem2^:=elem1^;
                                    elem1^:=temp_arbre^;
                                    dispose(temp_arbre);
                                    alterne_cellule := true;
                               end;
                      end
          end
End;
```

```
rocedure Tri_Liste (Liste : PListe);

var elem_N, elem_tri, elem_1, elem_N_1 : PListe;

egin
     If Liste^.suivant <> nil
        then begin
               elem_tri := liste;
               elem_n := nil;
               while (elem_tri <> elem_n) do
                 begin
                   elem_1 := nil;
                   if (elem_tri <> nil)
                      then begin
                             while ((elem_tri <> elem_N) and
                                    (elem_tri^.suivant <>nil))do
                               begin
                                 if Alterne_Cellule(elem_tri^.elem,
                                                    elem_tri^.suivant^.elem)
                                    then begin
                                           elem_N_1 := elem_tri;
                                           if (elem_1 = nil)
                                              then elem_1 := elem_tri
                                         end;
                                    elem_tri := elem_Tri^.suivant
                               end;
                             elem_n := elem_n_1;
                             elem_n_1 := liste
                           end
                      else elem_n := liste;
                   elem_tri := liste
                 end
             end
nd;
{ ---------------------- Fin Tri_Liste ---------------------------------- }

rocedure Supprime_Liste (Liste : PListe);

egin
     If  (liste^.suivant <> nil)
         then Supprime_Liste (liste^.suivant);
     dispose (liste);
nd;
{ ---------------------- Fin Supprime_Liste ------------------------------ }


EGIN
     If ((Arbre^.Categorie_Init = Plus) or (Arbre^.Categorie_Init = Fois))
        then begin
                  liste:= Cree_Liste(Arbre^.categorie_init,Arbre);
                  Tri_Liste(liste);
                  Supprime_Liste(liste)
             end;
ND;
 ***************************************************************;

Function Comparaison_Semantique (Arbre1,Arbre2 : PArbre) : Boolean;

ar egal : boolean;

Begin
     Comparaison_Semantique := false;
     If (Arbre1^.categorie_init = Arbre2^.categorie_init)
        then begin
               Case Arbre1^.categorie_init of

                    Inc, Nbre :
                       if compare (Arbre1,Arbre2,egal)and ega1
                          then Comparaison_Semantique:=true;

                    Plus, Fois :
                       if (   Comparaison_Semantique(Arbre1^.Poperand1,
                                                     Arbre2^.POperand1)
                          and Comparaison_Semantique(Arbre1^.Poperand2,
                                                     Arbre2^.POperand2))
                          or (   Comparaison_Semantique(Arbre1^.Poperand1,
                                                        Arbre2^.POperand2)
                             and Comparaison_Semantique(Arbre1^.Poperand2,
                                                        Arbre2^.POperand1))
```

```
                    then Comparaison_Semantique:=true;

        Moins, Divis, Puiss :
                  if (Comparaison_Semantique(Arbre1^.Poperand1,
                                             Arbre2^.POperand1)
                  and Comparaison_Semantique(Arbre1^.Poperand2,
                                             Arbre2^.POperand2))
                    then Comparaison_Semantique:=true;

            End;
          end;
End;


{********************  Fin Associativité ************************************}


procedure Commutativite (var Arbre : PArbre; var Msg : PChar;
                         var Erreur : boolean);

Var sous_arbre : PArbre;

begin
    Arbre := Neutre(Arbre);
    Case Arbre^.Categorie_Init of
      Plus .. Divis : begin
                      Commutativite(Arbre^.POperand2,Msg,erreur);
                      Commutativite(Arbre^.POperand1,Msg,erreur);
                      Ramene_Operateur_A_gauche
                           (Arbre,Msg,erreur);
                      Associativite(Arbre);

                    end;
            Puiss : begin
                      Commutativite(Arbre^.POperand2,Msg,erreur);
                      Commutativite(Arbre^.POperand1,Msg,erreur);
                      If (Arbre^.POperand2^.Categorie_Init = Nbre)
                         then begin
                              { <Facteur> ^0 => 1 }
                              if (Arbre^.POperand2^.Val_nbre.numerateur = 0)
                                  then begin
                                       new(sous_arbre);
                                       sous_arbre^.Categorie_Init := Nbre;
                                       sous_arbre^.Val_nbre.numerateur:=1;
                                       sous_arbre^.Val_nbre.denominateur:=1;
                                       sous_arbre^.suivant := nil;
                                       Supprime_noeud (Arbre);
                                       Arbre:=sous_arbre
                                       end
                                  else begin
                                         { <Facteur> ^ n/m où m <> 1 }
                                    if (Arbre^.Poperand2^.Val_nbre.denominateur
                                        <> 1)
                                          then begin
                                                Msg := 'Dois-tu vraiment '
                                                 +'calculer une racine ?';
                                                 Erreur := true;
                                                 Exit
                                               end
                                          else begin
                                                 { <Facteur> ^ -1 => 1/<Facteur> }
                                      if (- Arbre^.POperand2^.Val_Nbre.numerateur
                                       = Arbre^.POperand2^.Val_Nbre.denominateur)
                                      then begin
                                       sous_arbre := Arbre^.POperand1;
                                       Arbre^.POperand1 := Arbre^.POperand2;
                                       Arbre^.POperand1^.Val_nbre.numerateur := 1;
                                       Arbre^.POperand1^.Val_nbre.denominateur := 1;
                                       Arbre^.POperand2 := sous_arbre;
                                       Arbre^.Categorie_Init := Divis;
                                       Commutativite(Arbre,Msg,erreur)
                                           end
                                      else begin
                                                  {<Inc> ^ <Nbre>}
                                      if (Arbre^.POperand1^.Categorie_Init = Inc)
                                          then begin
                                       OpFraction_Puiss(Arbre^.POperand1^.coefficient,
                                             Arbre^.POperand2^.Val_Nbre,Msg,Erreur);
                                       If Erreur then Exit;
                                       Arbre^.POperand1^.Degre
```

```
                                    :=(Arbre^.POperand1^.degre
                                *(Trunc(Arbre^.POperand2^.Val_Nbre.numerateur)))
                             div (Trunc(Arbre^.POperand2^.Val_Nbre.denominateur));
                                if (Arbre^.POperand1^.degre = 0)
                                   then Arbre^.POperand1
                                       :=(Transforme_Inconnue_en_Nombre
                                                        (Arbre^.POperand1))
                                   else begin
                                   if ((Arbre^.POperand1^.degre <> 1)
                                   and (Arbre^.POperand1^.degre <> -1))
                                   then begin
                                       Msg := 'Nous sommes en train de '
                                             +'résoudre un problème du '
                                             +'premier degré(ax+b =0). '
                                             +'Essaie de ne pas élever '
                                             +'l''inconnue à une '
                                             +'puissance différente de '
                                             +'-1,0 ou de 1.';
                                       Erreur := true;
                                       Exit
                                         end;
                                        end;
                                     Supprime_Noeud(Arbre^.POperand2);
                                     sous_Arbre:=Arbre^.POperand1;
                                     dispose(Arbre);
                                     Arbre:=sous_arbre
                                     end;
                                                          end;
                                                 end;

                                                  end;
                                        end;
                              end;
    End;
nd;
 ---------------- Fin Commutativité ---------------------------------- }


unction Transforme_en_Plus (arbre : Parbre; var Msg : PChar;
                            var erreur : boolean) : Parbre;

Var mult,ptr_arbre,avance : Parbre;
    descend : boolean;
    prec    : Parbre;
    sommet  : boolean;


rocedure multiplie (var elem1 : PArbre; elem2 : Parbre;var Msg : PChar;
                    var erreur : boolean);

ar tampon : Tfraction;

Begin
     If elem2^.categorie_init = Inc
        then begin
             if elem1^.categorie_init = nbre
                then begin
                     tampon := elem1^.val_nbre;
                     Opfraction_Fois (tampon,elem2^.coefficient);
                     elem1:=elem2;
                     elem1^.coefficient:=tampon
                     end
                else begin
                     Opfraction_fois (elem1^.coefficient,elem2^.coefficient);
                     elem1^.degre := elem1^.degre + elem2^.degre;
                     If elem1^.degre = 0
                        then Transforme_Inconnue_en_Nombre (elem1)
                        else begin
                             if (elem1^.degre <>1) and (elem1^.degre <>-1)
                                then begin
                                     Msg := 'Nous sommmes en train de résoudre'
                                           +' un problème du premier degré '
                                           +'(ax+b=0). Essaie de ne pas '
                                           +'multiplier l''inconnue par '
                                           +'elle-même.';
                                     Erreur:= true;
                                     end;
                             end;
                     end
             end
```

```
      else begin
           if elem1^.categorie_init = nbre
              then Opfraction_Fois (elem1^.val_nbre,elem2^.val_nbre)
              else Opfraction_Fois (elem1^.coefficient,elem2^.val_nbre)
           end;
nd;
 ---------------------- Fin multiplie ------------------------------------ }


 ---------------------- Debut Transforme_en_plus ------------------------- }
egin
   if (Arbre^.categorie_init <> Inc) and (Arbre^.categorie_init <> Nbre)
   then begin
    descend := true;
    ptr_Arbre := Arbre;
    sommet := true;
    while descend do
      begin
           prec:=nil;
           while (ptr_Arbre^.categorie_init = Plus) do
             begin
                  prec:=ptr_arbre;
                  ptr_arbre:=ptr_arbre^.Poperand1;
                  sommet :=false;
             end;
           If sommet then Arbre := ptr_arbre^.Poperand1;
           If (ptr_Arbre^.categorie_init = Fois)
              then begin
                   mult:=ptr_arbre^.Poperand2;
                   avance :=ptr_arbre^.Poperand1;
                   case avance^.categorie_init of
              Inc,Nbre : begin
                         multiplie (avance^.Poperand1,mult,Msg,erreur);
                         descend:=false;
                         end;
                  Fois : multiplie (avance^.Poperand2,mult,Msg,erreur);
                  Plus : begin
                         while avance^.categorie_init = Plus do
                           begin
                                case avance^.Poperand1^.categorie_init of
                            Nbre,Inc : begin
                                       multiplie (avance^.Poperand1,mult,Msg,
                                                  erreur);
                                       multiplie (avance^.Poperand2,mult,Msg,
                                                  erreur);
                                       descend := false
                                       end;
                                  Fois : begin
                                       multiplie (avance^.Poperand2,mult,Msg,
                                                  erreur);
                                       multiplie (avance^.POperand1^.Poperand2,
                                                  mult,Msg,erreur);
                                       end;
                                  Plus : multiplie (avance^.Poperand2,mult,Msg,
                                                  erreur);
                                  end;
                                  avance := avance^.Poperand1;
                           end;
                         end;
                   end;
                   If prec <> nil
                   then prec^.Poperand1 := ptr_arbre^.Poperand1;
                   dispose (mult);
                   dispose (ptr_arbre);
                   ptr_arbre := avance
                   end
              else descend := false;
      end;
        end;
     Transforme_en_Plus:=Arbre;
nd;
 ----------------- Fin Transforme_plus ------------------------------------ }


Procedure Analyseur_Semantique (Var Arbre : Parbre; Arbre_Equation : Tab_membres;
ar msg : Pchar; var erreur : boolean;var equ_canonique :Tequ_canon; inconnue : char;
ar solution : Tfraction);

Var Cpt_inc : integer;
```

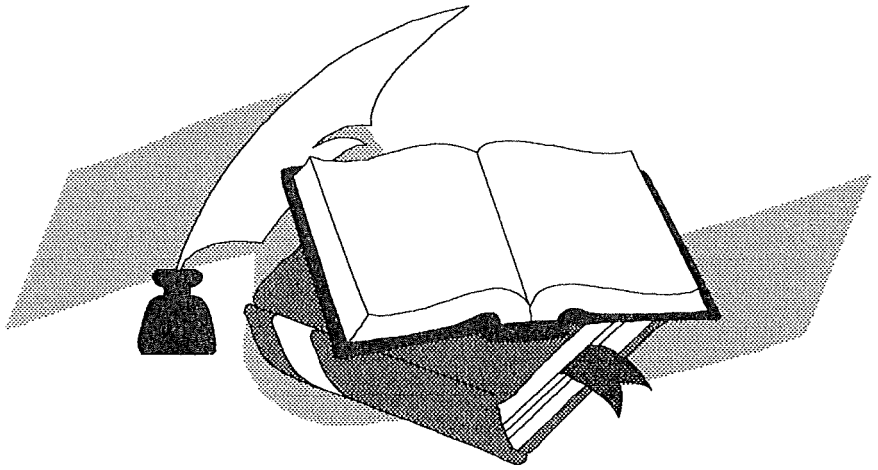
```
egin
  Arbre := Un_seul_Membre(Arbre_Equation);
  Commutativite (Arbre,Message,Erreur);
  If not Erreur
     then begin
          Cpt_Inc := 0;
          Lit_Membre(Arbre,inconnue,cpt_inc);
          if (Cpt_Inc < 1)
             then Msg:= 'Ce n''est pas une équation. Il n''y a pas d''inconnue.'
             else begin
                  lpile:=1;
                  Construit_pile(p,lpile,Arbre);
                  Msg :='Equation canonique : ';
                  Recherche_Equation_canonique
                 (equ_canonique,message,erreur,tampon,pilev);
                  if not erreur
                          solution_equation (solution,message,equ_canonique,
                          erreur);
                          end
                end;
          end;
nd;
  ----------------- Fin Analyseur_Semantique ---------------------------------- }


rocedure Compare_Equation (sol1,sol2 : Tfraction;
                           equ_canon1,equ_canon2 :TEqu_canon;
                           Arbre1,Arbre2 : PArbre);

ar egal : boolean;
   message : string;
   erreur  : boolean;
   liste1, liste2 : Pliste;

unction Cree_Liste_Feuilles (Arbre : PArbre) :PListe;

Var Cellule : Pliste;

egin
    New(Cellule);
    If (Arbre^.categorie_init = Inc) or (Arbre^.categorie_init = Nbre)
    then begin
         Cellule^.elem := Arbre;
         Cellule^.suivant := nil
         end
    else begin
         Cellule^.elem := Arbre^.Poperand2;
         Cellule^.suivant := Cree_Liste_Feuilles (Arbre^.Poperand1);
         end;
    Cree_Liste_Feuilles := Cellule
nd;


unction Compare_Cellule (Liste1, Liste2 : PListe) : boolean;

ar stop : boolean;
   fin  : boolean;
   egal : boolean;
   avance : PListe;
   Prec   : Pliste;
   fract1,fract2 : real;

egin
  Stop := false;
  While not Stop do
    begin
      fin := false;
      avance := Liste1;
      prec := liste1;
      While not fin do
        begin
          FractREAL(Liste2^.elem,avance^.elem,fract1,fract2);
          If abs(fract1) <> abs(fract2)
          then begin
               egal:=false;
               if avance^.suivant <>nil
                  then begin
                       prec := avance;
                        avance:=avance^.suivant;
```

```
                    end
                  else fin:=true
                end
          else begin
               fin:=true;
               egal:=true
              end;
        end;
     if egal
        then begin
             If avance^.suivant <>nil
                then begin
                     if prec <>avance
                        then begin
                             prec^.suivant :=avance^.suivant;
                             dispose(avance)
                             end
                        else liste1:=liste1^.suivant
                     end
                else begin
                     if prec <> avance
                        then prec^.suivant := nil
                        else begin
                             stop:=true;
                             liste1^.suivant:=nil
                             end;
                     end;
             end
        else begin
             stop:=true;
             With liste2^.elem^ do
              begin
                if categorie_init = Inc
                   then begin
                        Msg := 'Le terme contenant l''inconnue '+
                        abs(round(cofacteur.numerateur))+'/'+round(cofacteur.denominateur)+
                         ' x'+degre+ 'n''est pas une donnée de la première équation.'
                          end
                          else begin
                                Msg := 'Le nombre '+abs(round(val_nbre.numerateur))+'/'+
                                round(val_nbre.denominateur)+'n''est pas une donnée de'
                                + 'la première équation.';
                                end;
              end;
              end;
     If not stop and (liste2^.suivant <> nil)
        then liste2:=liste2^.suivant
        else stop:= true;
   end;
  if egal and (liste1^.suivant <> nil)
     then begin
          With liste1^.elem^ do
           begin
             if categorie_init = Inc
                then begin
                     Msg :='Le terme contenant l''inconnue '+
                        abs(round(cofacteur.numerateur))+'/'+round(cofacteur.denominateur),
                         ' x'+degre+' est un élément de la première équation que tu as oublié.'
                          end
                          else begin
                                Msg :='Le nombre '+abs(round(val_nbre.numerateur))+'/'+
                                round(val_nbre.denominateur)+'est un élément de'
                                + 'la première équation que tu as oublié.');
                                end;
              end;
            end;
  Compare_Cellule:=egal
nd;

{ ------------------------- Debut Compare_Equation ------------------------- }
begin
    erreur:=false;
    msg :='';
    If (sol1.numerateur = sol2.numerateur) and (sol1.denominateur = sol2.denominateur)
    then begin
         if Comparaison_Semantique (Arbre1,Arbre2)
            then Msg := 'Les équations sont sémantiquement équivalentes.'
         else begin
              Arbre1 :=Transforme_en_Plus (arbre1,Msg,erreur);
              if not erreur
```

```
                then begin
                     Associativite (Arbre1);
                     Arbre2 :=Transforme_en_Plus (arbre2,Message,erreur);
                     if erreur then writeln(message)
                     else begin
                     Associativite(Arbre2);
                     if Comparaison_Semantique (Arbre1,Arbre2)
                     or (Arbre2^.categorie_init = inc)
                     or (Arbre2^.categorie_init = nbre)
                        then Msg :='Les équations sont sémantiquement équivalentes.'
                        else begin
                               if (Arbre2^.Poperand1^.categorie_init = inc)
                               or (Arbre2^.Poperand1^.categorie_init = nbre)
                               then Msg :='Les équations sont sémantiquement équivalentes.'
                               else begin
                               Liste1:=Cree_Liste_Feuilles (Arbre1);
                               Liste2:=Cree_Liste_Feuilles (Arbre2);
                               If not Compare_Cellule(liste1,liste2)
                               then Msg :='Les équations ne sont pas sémantiquement équivalentes'
                               else Msg :='Les équations sont sémantiquement équivalentes'
                                   end;
                               end;
                          end;
                     end
                end
          end
     else begin
          Msg := 'Les solutions de ces deux équations sont différentes.'
           + 'Les équations ne sont pas sémantiquement équivalentes.'
          end;
nd;
 ---------------------------- Compare_Equation ---------------------------- }

EGIN
ND.
```

Annexe 2

Le manuel d'utilisation



Introduction

La disquette fournie avec ce mémoire contient différents programmes de test.

- TSIMPLIF.EXE Programme permettant de simplifier une fraction
- TOPFRACT.EXE Programme permettant de réaliser des opérations avec des fractions
- TSYNTAXE.EXE Programme permettant de vérifier si une expression entrée au clavier est une équation mathématique syntaxiquement correcte
- TCANON.EXE Programme permettant de transformer une équation sous sa forme canonique et de calculer la solution d'une équation
- TSEMAN.EXE Programme permettant de vérifier si une équation A, entrée au clavier, est l'étape suivante de la résolution de l'équation B, elle aussi entrée au clavier
- TPLUS.EXE Programme permettant de ramener une équation composée d'opérateurs différents à une équation ne contenant que des opérateurs d'addition

Pour exécuter ces différents programmes, vous devez disposer de Windows 3.x. Placez la disquette dans le lecteur A: et lancez Microsoft Windows. Pour faire démarrer chaque programme, placez-vous dans le gestionnaire de programmes. Choisissez l'item de menu "Fichier/Exécuter" et tapez la commande a:\Nom.EXE où 'Nom' représente un nom de fichier

La disquette du programme 'Resolver' n'est pas fournie avec ce mémoire. Ce type de logiciel requiert une version correcte du moteur de base de données Borland. En effet, lors de l'installation d'une application de base de données, il faut installer un BDE redistribuable avec son propre utilitaire d'installation, que nous pouvons redistribuer gratuitement avec notre application.

Le contrat de licence de Delphi requiert que tous les fichiers du BDE redistribuable soient mis à la disposition des utilisateurs de l'application. Ceci permet aux utilisateurs d'installer la nouvelle version du BDE pour Delphi sans interférer avec des applications Paradox et dBASE existantes.

Installation du logiciel 'RESOLVER'

Pour installer le logiciel 'Resolver', placez la disquette dans le lecteur A:. Ensuite, tapez la commande A : Install.Bat. Celle-ci va décompresser les fichiers contenus sur votre disquette et va les copier sur votre disque dur dans le répertoire 'Resolver'.

Lancement de 'Resolver'

Resolver contient un module destiné au professeur et un autre module destiné à l'élève.

 Pour lancer le module du professeur, lancez Microsoft Windows. Ensuite, placez-vous dans le gestionnaire de programmes. Choisissez l'item de menu "Fichier/Exécuter" et tapez la commande c:\resolver\Prof\Prof.EXE

 Pour lancer le module de l'élève, lancez Microsoft Windows. Ensuite, placez-vous dans le gestionnaire de programmes. Choisissez l'item de menu "Fichier/Exécuter" et tapez la commande c:\resolver\eleve\eleve.EXE

Les écrans

Lorsque vous lancez le module du professeur, le programme se présente, après quelques instants, par l'écran FEN 1 (Cfr. 4.5.1, page 76). L'utilisateur qui a normalement accès à ce programme connaît le code secret. Vous devez faire défiler le curseur jusqu'à l'obtention de ce numéro. Ensuite, validez-le en appuyant sur le bouton de commande *'OK'*. Si le code est incorrect, l'application se termine.

Si le numéro est correct, un nouvel écran apparaît (Cfr. 4.5.2, page 77).

La barre de titre

Au sommet de l'écran, vous voyez la barre de titre commune à tous les programmes fonctionnant sous Windows, dans laquelle figure le nom de l'application courante. En ce qui concerne le module du professeur, ce titre est composé du programme 'Resolver' et du nom du module.

A l'extrémité droite de la barre de titre vous voyez un bouton avec une croix (Windows 95). Si vous cliquez sur ce symbole à l'aide de la souris, l'application se terminera.

La barre des menus

Au-dessous de la barre de titre se trouve la barre des menus, dans laquelle figurent les noms des menus de Resolver

Le menu Exercice

Ce menu regroupe les commandes suivantes : *'Imprimer ...'*, *'Configuration de l'impression ...'* et *'Quitter'*.

Le menu Edition

Ce menu se compose des commandes *'Couper'*, *'Coller'*, *'Copier'* et *'Supprimer'* qui se retrouvent dans pratiquement toutes les applications Windows.

Le menu Relation

Ce menu ne contient qu'un seul item *'Etablir ...'*. Il permet d'activer la boîte de dialogue FEN 2 (cfr. 4.5.3, p 77).

Le menu Aide

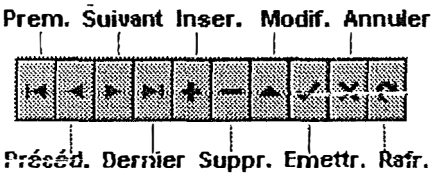
Le menu d'aide contient un index et des informations sur Resolver.

Pour le module du [illegible] la barre des menus contient un menu supplémentaire 'Editeur' qui permet d'ouvrir la boîte de dialogue FEN 5 (Cfr. 4.5.5, page 78).

La barre d'outils

Il s'agit d'un composant TDBNavigator (composant navigateur de base de données) qui est utilisé pour naviguer dans les données d'une table ou d'une requête d'une base de données et pour accomplir des opérations sur ces données telles que l'insertion d'un enregistrement vide ou l'émission d'un enregistrement. Il est utilisé conjointement avec les contrôles sensibles aux données tels que les grilles de données pour permettre la modification et l'affichage des données.

Le navigateur de base de données est constitué de plusieurs boutons.



Lorsque vous choisissez l'un des boutons du navigateur, l'action voulue se déclenche sur l'ensemble des données auquel le navigateur est lié. Si, par exemple, vous cliquez sur le bouton *'Insérer'*, un enregistrement vide est inséré dans l'ensemble des données.

Le tableau suivant décrit les boutons du navigateur :

BOUTON	ROLE
Premier	Le premier enregistrement de l'ensemble des données devient l'enregistrement courant, les boutons Premier et Précédent sont désactivés et les boutons Suivant et Dernier sont activés
Précédent	L'enregistrement précédent devient l'enregistrement courant, les boutons *Dernier* et *Suivant* sont activés
Suivant	L'enregistrement suivant devient l'enregistrement courant et les boutons *Premier* et *Précédent* sont activés
Dernier	Le dernier enregistrement de l'ensemble des données devient l'enregistrement courant, les boutons *Dernier* et *Suivant* sont désactivés et les boutons *Premier* et *Précédent* sont activés
Insérer	Insère un nouvel enregistrement qui précède l'enregistrement courant et bascule l'ensemble de données en mode insertion et modification
Supprimer	Supprime l'enregistrement courant et l'enregistrement suivant devient l'enregistrement courant
Modifier	Bascule l'ensemble des données en mode modification pour que l'enregistrement courant puisse être modifié
Emettre	Ecrit les modifications de l'enregistrement courant dans la base de données
Annuler	Annule les modifications dans l'enregistrement courant et restitue l'état de l'affichage de l'enregistrement tel qu'il était avant la modification, désactive les modes insertion et modification s'ils sont actifs
Rafraîchir	Réaffiche l'enregistrement courant de l'ensemble des données, provoquant ainsi la mise à jour de l'affichage de l'enregistrement sur la fiche

Utilisation de Resolver

Module du professeur - Création d'exercices

Entrer un énoncé

Vérifier que le bouton *'Modifier'* du *Navigateur* est enfoncé. Ensuite, il vous suffit de vous positionner dans le mémo situé en dessous du label *Enoncé*. Pour y accéder, vous pouvez utiliser les touches <Alt> <N>.

 Vous pouvez définir certains mots de l'énoncé du problème. Il vous suffit de le sélectionner à l'aide de la souris et d'appuyer sur le bouton de commande *'Définir'*. La boîte de dialogue *'Dictionnaire'* s'ouvre. Donnez la signification de ce mot et un exemple; vous pouvez ajouter d'autres mots en appuyant sur le bouton '+' du Navigateur.

Mettre le problème en équation

Vérifier que le bouton *'Modifier'* du *Navigateur* est enfoncé. Ensuite, il vous suffit de vous positionner dans la boîte d'édition située à coté du label *Equation*. Pour y accéder, vous pouvez utiliser les touches <Alt> <Q>. Ensuite, validez la mise en équation en appuyant sur le bouton *'Ok'* de la boîte de regroupement *'Mise en équation et résolution'*. Le logiciel résout l'équation et en donne la solution. Si la solution est un cas extrême (cfr. page 33). Le logiciel affichera un commentaire dans le mémo de la boîte de regroupement *'solution du problème'*. Ce mémo peut être modifié par l'utilisateur. Il vous suffit de vous positionner dans la boîte d'édition située en dessous du label *Commentaires*. Pour y accéder, vous pouvez utiliser les touches <Alt> <M>.

Proposer des inconnues

L'utilisateur entre au moins une proposition pour l'inconnue dans le mémo situé en dessous du label *Propositions*. Pour y accéder, vous pouvez utiliser les touches <Alt> <P>. Ensuite, il faut sélectionner à l'aide de la souris l'inconnue du problème. Pour valider la réponse, il suffit d'appuyer sur le bouton *'Sélectionner'*.

Etablir des relations entre l'énoncé et l'équation

Choisissez le menu *'Relation'* et cliquez sur l'item *'Etablir ...'*. La boîte de dialogue amodale s'ouvre. Sélectionnez dans l'énoncé un ou plusieurs mots (le bouton *'MultiSéletion'* doit être enfoncé si vous voulez sélectionner plusieurs mots). Ensuite sélectionner dans l'équation ce qui y correspond.

Pour chaque nouvelle relation, appuyez sur le bouton '-' de la barre d'outils de la boîte de dialogue *'Relation entre l'énoncé et l'équation'*. Ensuite, Vous pouvez associer à la relation du commentaire.

Tant que la boîte est ouverte, les sélections dans l'énoncé et dans l'équation sont considérées comme des relations que l'utilisateur établit entre l'énoncé et l'équation.

Utilisation de Resolver

Module de l'élève - Création d'exercices

Comprendre l'énoncé

L'élève peut obtenir des informations sur certains mots de l'énoncé si ceux-ci sont dans la base de données. Il suffit de sélectionner un mot dans l'énoncé à l'aide de la souris et de cliquer sur le bouton de commande *'Définir'*

Choisir l'inconnue

L'élève choisit dans la liste déroulante, parmi les propositions, l'inconnue à l'aide de la souris et valide sa réponse en appuyant sur le bouton *'Sélectionner'*.

Etablir des relations et mettre le problème en équation (cfr. Le module du professeur)

Après avoir mis le problème en équation, il faut encore résoudre cette équation. Pour ce faire, l'élève dispose d'un mémo où il peut décrire toutes les étapes pour résoudre cette équation. Il suffit d'activer la boîte de dialogue *'Editeur d'équations'* en choisissant dans le menu *'Editeur'*, l'item *'Résoudre'*. Ensuite, dès qu'il a trouvé la solution de l'équation, il peut entrer sa réponse dans la boîte d'édition *'solution de l'équation'*. Si la solution est correcte, un commentaire sur la solution du problème sera affiché.

Quand il a terminé l'exercice, il valide son travail appuyant sur le bouton *'OK'* situé en bas à droite dans la fenêtre principale.